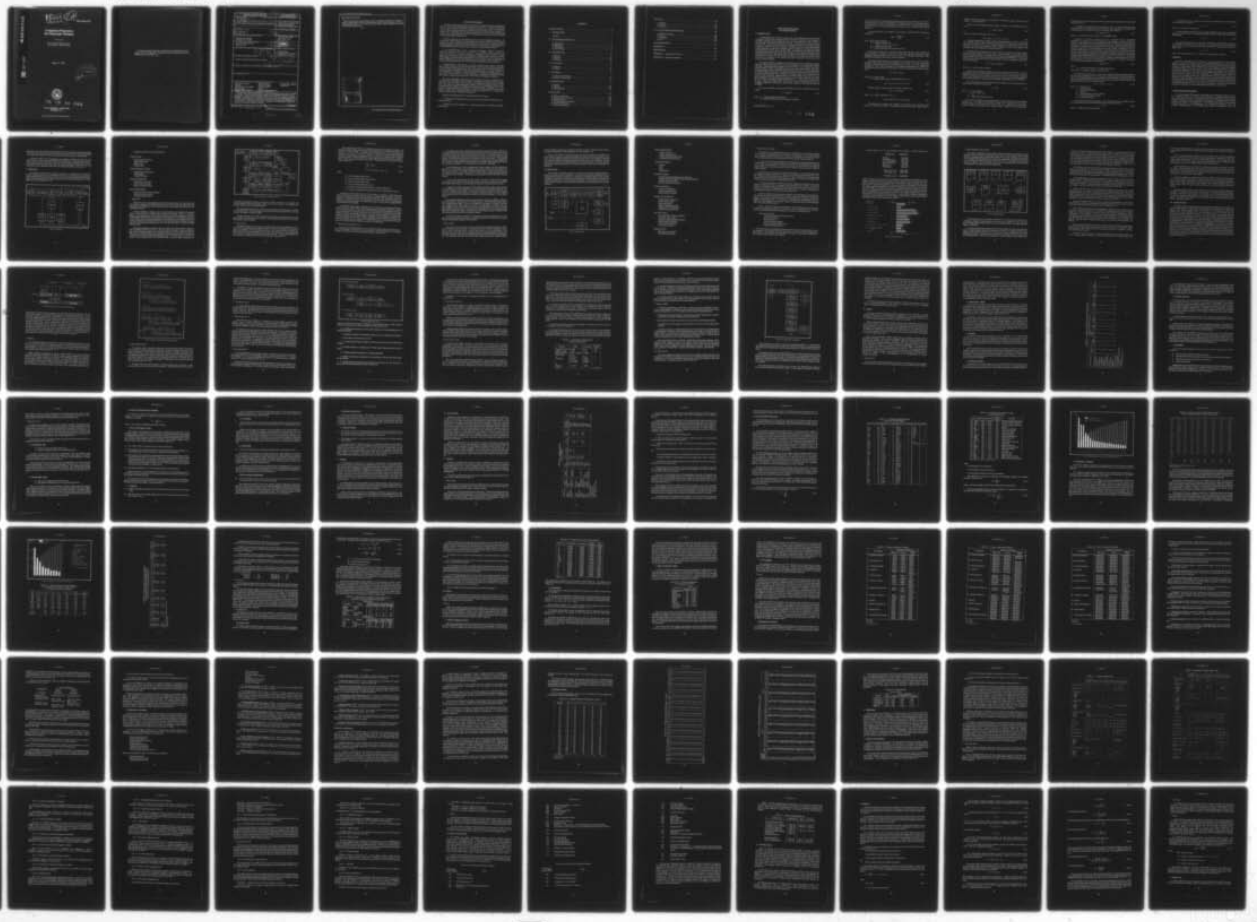
AD-A060 325 NAVAL RESEARCH LAB WASHINGTON D C F/G 9/2
COMPUTERS/PROCESSORS (FOR ELECTRONIC WARFARE).(U)
AUG 78 L W LEMLEY

UNCLASSIFIED NRL-8247 NL

1 OF 2
AD
A060 325



AD A060325

DDC FILE COPY

LEVEL II

NRL Report 8247

Computers/Processors (for Electronic Warfare)

L. W. LEMLEY

Electronic Support Measures Branch
Tactical Electronic Warfare Division

August 15, 1978

DDC
RECEIVED
OCT 26 1978
F

78 10 16 004

NAVAL RESEARCH LABORATORY
Washington, D.C.

Approved for public release; distribution unlimited.

The Naval Research Laboratory is authorized to reproduce and sell copyrighted items in this document. Permission for further reproduction must be obtained from the copyright owner.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NRL Report 8247
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): COMPUTER PROCESSORS (FOR ELECTRONIC WARFARE)
5. TYPE OF REPORT & PERIOD COVERED: Interim report on one phase of a continuing NRL Problem.
6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): Leo W. Lemley
8. CONTRACT OR GRANT NUMBER(s): NRL-8247

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Research Laboratory
Washington, D.C. 20375

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
NRL Problem R16-47
WF 12-111-706

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Air Systems Command
Washington, D.C.

12. REPORT DATE
August 1978

13. NUMBER OF PAGES
176

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Architecture, computer	Electronic Warfare	Software (EW), computer
Computer	Evaluation, computer	Specification
Computer analysis, quantitative	Future, computers	
Cost, computer	Processor	
Electronic Support Measures	Selection, computer/processor	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Computers/processors (for EW/ESM) have long eluded quantitative definition both in hardware and software. This work provides quantitative factors by which (EW/ESM) requirements may be translated into computer/processor evaluation, specification, and selection. Hardware and software (EW/ESM) are addressed as well as architecture, costs, and the future of computers/processors. The CFA/MCF work to quantify the software aspects of computers is translated into EW/ESM application and relevancy. Essentially this document is a handbook by which (EW/ESM) computer processors may be evaluated, specified, or selected.

(Continued)

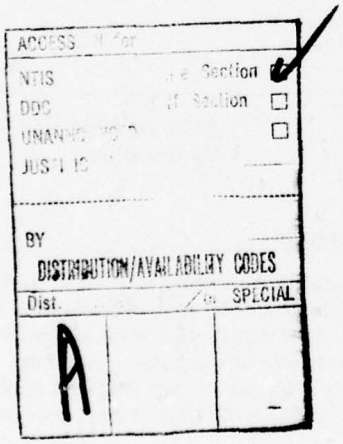
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

251 950

20 ABSTRACT (Continued)

While many factors have varying influence upon the (throughput) performance of a computer/processor, memory speed is pivotal to this performance, and developing technology is the basis of improving memory speed. In view of the high cost of software, it should be designed to be tolerant of technological improvements.



EXECUTIVE SUMMARY

This rather massive work represents a need to bound computer/processors in quantitative values. The work is EW/ESM-oriented because the requirement is to define EW/ESM subsystems (such as the computer/processor) in quantitative terms; however, much of the methodology and values derived are general computer/processor and general military applicable. In essence, this is a handbook of computer/processors although it lacks the overall completeness of a handbook. Some redundancy is written in deliberately to permit section access for specific needs, avoiding tedious cross-reference.

The EW community has long labored under a restricted throughput requirement. The "choke-point" of this throughput is the preprocessor that can only evaluate a limited number of emitters per second. This is true whether dealing with an electronic reconnaissance platform that must distinguish hostile/friendly emitters and their wartime changes in rapid response; or, the ECM/RWR equipment that must react to a threat in a high density environment.

The computer/processor, in turn, is limited by technology. Contrary to community trends, the *cost-effective EW processor is based upon well-established, software-supported, large-volume technology,* not on exotic new architecture. Throughput is directly related to memory speed and to some extent memory-speed-dependent microinstruction execution time. Most well-established, software-supported architectures are easily adapted to new technology for higher throughput without expensive software-architectural changes. Widely available architectures are competitive performers with EW custom-built architectures, and cost much less.

Computer/processor performance is primarily dependent upon memory technology. Speed, volume, weight, cost are all primarily functions of the speed-power product characteristic of the memory technology employed. The software support cost of a computer/processor is a function of the total investment in a particular computer/processor architecture. Within common technological boundaries, hardware architecture (including microprogramming) can account for a ± 43% variation in a computer/processor's (throughput) performance. Higher level, software architecture can account for ± 14% variation in throughput performance.

The future (1980-2000) computer/processors can plan on an order-of-magnitude (10×) cost reduction (per throughput) every ten years. Throughput will improve an order of magnitude (10×) every seven years, with a military lag over commercial of approximately seventeen years. The volume of computer/processors reduced an order of magnitude every five years. Computer/processor weight (per throughput) also reduces an order of magnitude (10×) every five years. These predictions are based upon a rather substantially based trend in component industry, twenty years of computer/processor experience, supported by an ongoing level research that permits these technology improvements.

Among the recommendations that chiefly emerge from this study are

1. Systems should be designed with existent, widely used processor architecture, not customized processors.

2. Software should be designed to be operationally tolerant of hardware technology improvements.

CONTENTS

Executive Summary iii

I. INTRODUCTION 1

Summary 5

II. PROCESSOR REQUIREMENTS 5

A. Environment 6
B. ESM System 7
C. Preprocessor 8
D. Main Processor 13

III. ARCHITECTURE 19

A. Hardware 20
B. Software 30
C. Summary 39

IV. EVALUATION 40

A. Hardware 40
B. Software 54

V. SOFTWARE 72

A. Software Tool Description 72
B. Software Tool Evaluation 82

VI. SPECIFICATION 85

A. Hardware 86
B. Software 102
C. Microinstruction 107

VII. SELECTION 108

A. Memory Access Time 108
B. Instruction Execution Time 109
C. Benchmark Tests 110
D. Architectural Element Weights 112
E. Tool Availability Index 113
F. Physical Characteristics 114

VIII. COST 116

A. Hardware 117
B. Software 122
C. Life Cycle 123

IX. FUTURE COMPUTER/PROCESSORS 126

A. Statistical Forecasting 126
B. Density 128
C. Throughput 135
D. Cost 138

ACKNOWLEDGMENT 140

REFERENCES 140

BIBLIOGRAPHY 141

APPENDIX A — Derivation of Table IV-9 149

APPENDIX B — Benchmark Algorithms 158

COMPUTERS/PROCESSORS
(for Electronic Warfare)

I. INTRODUCTION

A considerable amount of work has been carried out on (EW) processors in the interests of satisfying the high data throughput required in the EW environment. This problem has stimulated a significant amount of processor work even to the most basic research, currently in the technology of the Josephson-Effect Device (JED). Unfortunately, no work has been performed on the quantitative requirements of such processors, other than a superfluous attempt to support a particular processor technology. Solutions range from various highly efficient software algorithms, through various hardware processor architectures, new logic technologies, to new solid state research, such as the above JEDs. Although all of these approaches present captivating arguments in the solution of (EW) processors, an introspective, quantitative analysis of processors as related to EW indicates that the EW community might well examine closely these solutions as "virtual," and that the actual *cost-effective results indicate that an EW processor constructed of well established, software supported, large volume technology wins hands down over any unique technological approach.*

The extremely expensive, programmer-related software costs in the development of algorithms and software tools should result in a close examination of any proposed new processor technology. Established general architectures, such as PDP-11 or Intel 8080, have demonstrated an ability to adjust to technological advances such as high speed memories while retaining the advantages of their considerable investment in software tools. A further advantage of the familiar architecture approach is the general familiarity of programmers with these processors and their software tools. Such an approach is in marked contrast to the manufacturer who has built a better "mousetrap" processor solution to the EW problem, but only one or two company programmers know the machine well enough to use it. Studies carried out under NRL Problem 57R16-47 (NAVAIR) indicate technologically that, as long as the machines contain the same memory technology (MOS, bipolar, core), they are essentially equivalent in throughput. Regardless of architecture, memory speed matched to CPU determines real-time throughput.

The cost of a processor is closely correlated (0.902) to the total investment in that architecture;

$$C_{si} = 42\ B_2^{-0.15}, \tag{I-1}$$

where C_{si} = \$ cost per instruction (software)
B_2 = \$M total dollar investment in computer architecture,

Manuscript received July 17, 1978.

78 10 16 004

such that the larger the value of delivered hardware inventory, the smaller the cost of a (EW) processor system. In view of the adaptability of established processors to technology improvements (bipolar memories), and technologically free software tools, one must conclude that the introduction of a "new" fast processor architecture would be a cost- and performance-inefficient adventure.

As mentioned above, the throughput of any processor is directly related to the memory access time,

$$PRF = \frac{13264 N_c}{T_m^{2/3}}, \tag{I-2}$$

where PRF = pulses per second
N_c = number of parallel CPUs
(where $T_p > T_m$, $N_c = T_p/T_m$)
T_m = memory access time (μs)
T_p = average microinstruction execution time (μs).

This relation is based upon a minimum tracking algorithm, and as such represents an upper boundary for processors. There may be more basic algorithms that could increase throughput somewhat; however, the increase would not be a significant improvement. More sophisticated algorithms would result in reduced throughput.

The minimum boundary physical characteristics of a processor are also closely correlated to memory size, whether they be military qualified or commercial. Of course, the minimum boundary commercial characteristics outperform military values due to the ruggedization requirements. These values may change with technological improvements that are addressed in future processor projects; however, at present (ca. 1977) the required power for military processors is

$$Pwr = 36.3 + 4.8\ M, \tag{I-3}$$

where Pwr = power in watts
M = static memory capacity (16-bit kilowords) and $\pm 1\ \sigma$ is

$$5.4 + 4.5\ M < Pwr < 67.1 + 5.0\ M. \tag{I-4}$$

Processor weight, in the same manner, is related to memory size:

$$W_p = 3.7 + 0.69\ M, \tag{I-5}$$

where W_p = weight in pounds and $\pm 1\ \sigma$ is

$$-16.5 + 0.59\ M < W_p < 9.2 + 0.79\ M. \tag{I-6}$$

The appearance of a negative value (weight) for a low memory size is obviously invalid and due to the statistical method of analytically representing processor weight related to

memory size. However, when the total expression is examined, the upper 1σ limit allows a low memory — weight relationship.

Finally, the volume relationship also correlates to memory size (as defined above) as follows:

$$V_p = 369 + 7.67\ M, \tag{I-7}$$

where V_p = volume in cubic inches and $\pm 1\ \sigma$ is

$$187 + 6.29\ M < V_p < 551 + 9.05\ M. \tag{I-8}$$

The above relationships all apply to military standard experience (ca. 1977) in which generally airborne processors where analyzed. Airborne processors where chosen, since weight, size, power are more critical to this platform (satellites are grouped with airborne). The rationale is that performance designed to airborne requirements more nearly represents what the state of the art can accomplish in these characteristics.

Evaluating physical characteristics of more advanced commercial (or R&D) products becomes a bit more speculative as may be expected, since no attempt has been made to optimize their packaging. However, some insight may be achieved by comparing the pertinent factors that contribute to these physical characteristics at the commercial and R&D level. For instance, present commercially available static memories display an order of magnitude improvement over standard military;

$$Pwr = 0.334 + 0.0363\ M, \tag{I-9}$$

where M = memory capacity (kilobits).

The costs of acquiring and operating a processor have been particularly elusive. Only after considerable analysis of various references did a good concept of the costs of a processor emerge. There are three main elements of processor costs: (a) the initial hardware acquisition costs, (b) the software costs, and (c) the life-cycle costs. These costs may be used together for an up-to-date calculation of cost expectancy even though the base figures are ca. 1977/1976.

The initial acquisition costs of computer processor hardware are most generally related to the processor's instruction execution time:

$$C_{hi} = \frac{k}{T_p^g}, \tag{I-10}$$

where C_{hi} = cost in dollars
k = 6.3×10^4 for 1976
g = 0.4
T_p = instruction execution time, μs.

In the matter of software, cost per instruction is the most general manner of evaluating software costs. An analysis of EW programs indicates that a tracking preprocessor would minimally require 32 to 98 instruction steps per pulse word, depending upon architecture (series or parallel) and parameters (AOA, frequency, TOA). Analysis of a main EW program

indicates that the normally expected main EW computer software would involve some 15,000 steps of instruction.

As implied in the beginning of this introduction, there is a close association between software cost and the inventory (value) of the processor. Most references use a software "Tool Availability Index" as an indicator of the cost (per instruction) of software.

$$C_{si} = 1138\ (TAI)^{-1.07}, \tag{I-11}$$

where C_{si} = dollar per instruction and TAI = tool availability index.

Normally, it is not easy to assess the TAI for a processor without a considerable knowledge of the software tools that are available for that processor, not only by title but by content. A more powerful (and easily implemented) index of software costs (with equally high correlation) is the value of the total processor inventory in use. The latter criterion provides a good index of potential software costs based upon the rationale that the higher the inventory value, the lower the cost of programming due to the total number of processors in use, the software tools developed to use these processors, and a greater number of programmers familiar with the processor and the software tools. Again, good examples of this rationale are the ease and lower costs involved in the application of machines such as DEC's PDP-11 or Intel's 8080. Common programming languages do not comprise an exclusive answer, since a company's investment in its processor and a broad inventory of usages are more important factors to the cost of software. As such, the cost of software is highly correlated ($r = 0.902$) to the total dollar value of the basic installed computer/processor:

$$C_{si} = 42.34\ B_2^{-0.149} \tag{I-12}$$

where

B_2 = total dollar value ($M) of installed processor base.

This value of C_{si} is adjusted inflation for (and is current).

The final value, life cycle costs, is related to a number of factors, but primarily time (in years), of course, decides the life cycle costs. Again, a number of references were analyzed to provide the following guidelines as to the effects on this consideration. Aside from the time factor, number of units, initial hardware cost, and software initial costs provide the factors that determine the life cycle cost of a processor system:

$$C_{LC} = n_i\ di\ L_h\ C_{hi} + L_s\ S_{wi}, \tag{I-13}$$

where C_{LC} = life cycle cost,
n_i = number of units
d_i = quantity discount factor
L_h = life cycle
C_{hi} = initial acqusition cost/unit, hardware
L_s = life cycle cost factor, software
S_{wi} = initial software costs.

Certain factors require further definition. The software life cycle cost is based on experience in the support of the initial software base plus a time familiarity factor:

$$L_s = t^{0.384}, \tag{I-14}$$

where t = number of years from acquisition.

The initial software cost is based on the previously determined cost per instruction and the total number of instructions:

$$S_{wi} = C_{si} I, \tag{I-15}$$

where I = total number of instructions.

The initial hardware acquisition cost was previously defined in Eq. (I-10). The hardware life-cycle-cost factor is also a time dependent variable in which a time/learning experience has been factored:

$$L_{hi} = t^{0.777}. \tag{I-16}$$

The quantity discount factor d_i is the result of experience in the reduction of cost due to the purchase of a number of units on one order. It is less accurate for small numbers than larger numbers, but it is a factor that must be assessed where a system may be evaluated on quantity cost:

$$d_i = n_i^{-0.142}, \tag{I-17}$$

where cost reduction factor due to N_i, number of units purchased on one order.

SUMMARY

The above relationships are the first-order, executive guidelines for the evaluation specification and selection of (EW) processors. The detailed analytic support for these approximations is presented in the ensuing report along with the rationale. Since technology is rapidly advancing, in a final chapter some consideration is given to the predictive impact of commercial and R&D technology on processors in the outyears. These predictive factors are of necessity less confident, depending on their degree of development and potential. The procurement planner may desire to wait for the fruition of these potentialities, depending on the urgency of the need and the processor task to be accomplished. It is important, however, to be well informed and avoid claims without basis. Processors (EW) must be evaluated, specified, and selected on a firm technological, cost/performance basis. The purpose of this study and report is to provide those quantitative factors by which the procurement planner may demand such responsiveness from the vendor. To the vendor's advantage, he may know by this report to what procurement factors the vendor may be expected to respond. If the vendor cannot justify his product on this basis, then he will know why his processor is not competitive.

II. PROCESSOR REQUIREMENTS

A number of general EW processor requirements will be addressed in detail in this section. These requirements, in turn, will lay the groundwork for the ensuing sections on evaluation, specification, and selection. The emitter environment requirements have been analyzed enough so that pulse rates can be established for the processor. An ongoing analysis of emitter environment in depth is to be published later. The EW/ESM requirements are the systematic processor requirements. A third requirement is the preprocessor requirements; what is the function of the preprocessor in the EW/ESM system. Finally, the requirements of the main

processor are addressed. After the preprocessor has categorized (not identified) the emitters, the main processor must analyze, identify, and report the results to a user in time for a reaction if necessary.

A. Environment

A modern high-density EW signal environment involves approximately 1600 emitters that radiate some 1.2 million pulses per second (pps). While these figures are platform and system dependent, they do represent the magnitude of the ESM/EW processor problem. In Fig. II-1 the environment represented is the East-West German border, with platforms at various altitudes. Some features of this representation are listed below.

1. System sensitivities to -70 dBm are essentially altitude independent. Most operable altitudes will result in the same pulse density regardless of altitude at a particular system sensitivity.

2. Systems more sensitive than -70 dBm encounter environments that are both altitude and sensitivity dependent.

3. The environment that an ESM system will encounter above approximately -106 dBm is strictly altitude dependent, with system sensitivity playing a relatively minor role in pulse density.

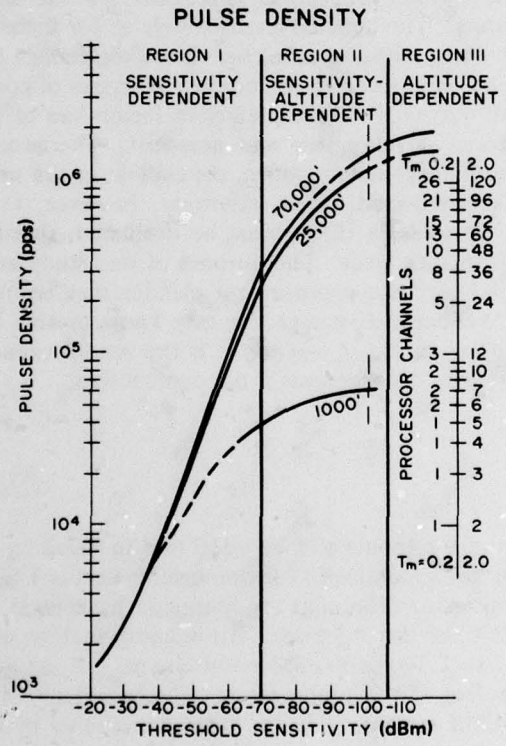


Fig. II-1—Representative high-density noncommunications environment

It is obvious, then, that the processor requirements of an EW/ESM system are dependent upon its sensitivity and altitude/horizon. It is equally obvious that the coverage of an EW/ESM system is dependent upon these same factors — sensitivity and altitude/horizor.

B. ESM System

A functional block diagram of the processing requirements of a postulated ESM System is shown in Fig. II-2. This system is designed to continuously monitor the signal environment and to intercept, identify, and derive a direction of arrival (or location) for all emitters of interest to the system. System performance is measured in terms of timely identification of emitters on a prioritized basis. The output of the ESM System would be sent to a data recorder, a video display, or a threat alarm, or it could be used to automatically activate an ECM System.

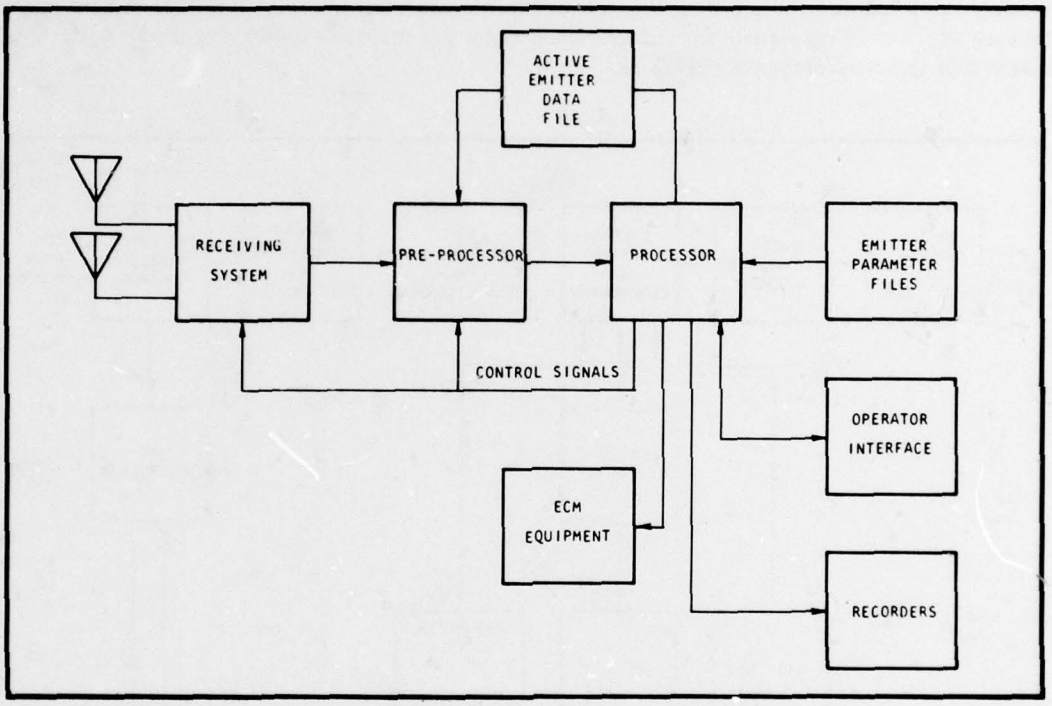


Fig. II-2—Typical ESM system

The modernized ESM antenna and receiving system consists of digitally controlled and tuned equipment capable of covering the entire frequency spectrum of interest with a high probability of intercept and providing at its output the individual detected radar pulses and sufficient information to calculate their direction of arrival (DOA).

The radio frequency (RF) receiving system is required to monitor specified frequency and spatial domains continuously. However, performance tradeoffs such as sensitivity, resolution, and dynamic range may require that a time-ordered sampling in the frequency and spatial

domains be used. Thus, frequency-scanning receivers or spatial-scanning antennas are generally a part of the receiving system, and the resultant output of the system is a time-ordered sample of the signal environment that must be processed and reconstructed by the ESM processor.

Numerous antenna/receiver configurations are employed in ESM systems. However, the output of each is similar, consisting of digital words indicating the average frequency, angle of arrival (AOA), and the time of arrival (TOA) of the leading edge of each signal pulse. The system may also output the actual received pulse for further processing, in which case other pulse characteristics such as amplitude, pulse width, and instantaneous phase can be derived.

C. Preprocessor

The parametric data from the receiving system go to a preprocessor for preliminary analysis. The main function of the preprocessor is to filter out the emitters that have been previously identified and the emitters that are of no interest from the incoming data stream and to prepare the remaining signals for further analysis by the main computer. Figure II-3 is a block diagram of typical preprocessor. [1,2,3]

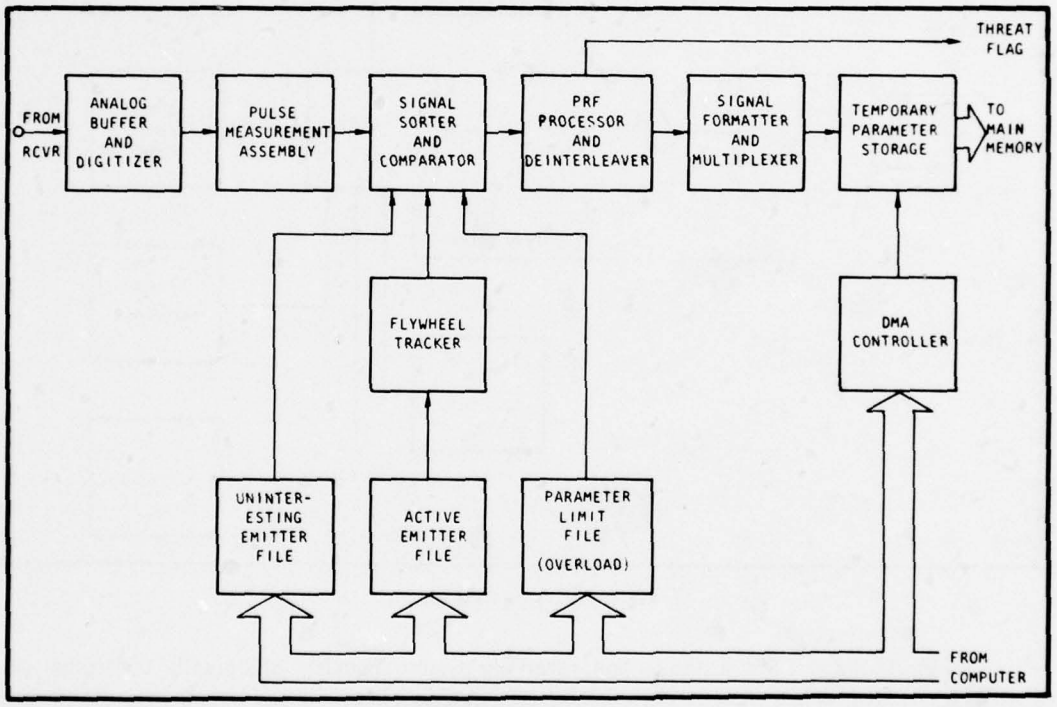


Fig. II-3—Preprocessor

Preprocessor functions are summarized below.

Prepare the Data

Digitize data from receiver
Measure pulse width
Measure TOA
Calculate AOA

Perform Preliminary Analysis

Deinterleave incoming pulses
Calculate PRF
Analyze PRF agility

Sort the Incoming Pulses

Predict TOA of next pulse
Track the active emitters
Blank the Friendly emitters

Load the Main Memory

Format the data for further processing
Interrupt the main computer
Write under DMA control

1. Signal Sorter

Figure II-4 shows a typical signal sorter and the associated circuits. The data stream from the receiver is compared on a pulse-to-pulse basis with a series of data files. Data words that match each file are deleted from the data stream to reduce the throughput requirements of each succeeding comparator.

The first comparison is based on a file of emitters under active tracking by the flywheel tracker. Each data word is compared with the expected parametric values (i.e., frequency, DOA, and the expected TOA) of each emitter being actively tracked. Comparison matches of tolerance limits for each emitter are calculated for each parameter by the flywheel tracker, based on the variation of the previous measurements for the parameter. If the comparison results in a match, the signal pulse is deleted from the data stream and the differences between the actual data and the nominal signal parameters are sent to the flywheel tracker to update the tracked emitter data file.

The second comparison is based on data for emitters fully characterized for frequency, DOA, and PRI, but which have lost track. This may be due to emitter or system antenna scanning, system frequency scanning, or a temporary change in the emitter's mode of operation. The data in the nontracked emitter file corresponds to emitters that are expected to be received and that match the system antenna (and frequency) scan. If a match occurs, the TOA of the intercept is used, along with the known PRI information, to achieve a rapid "reacquisition" of

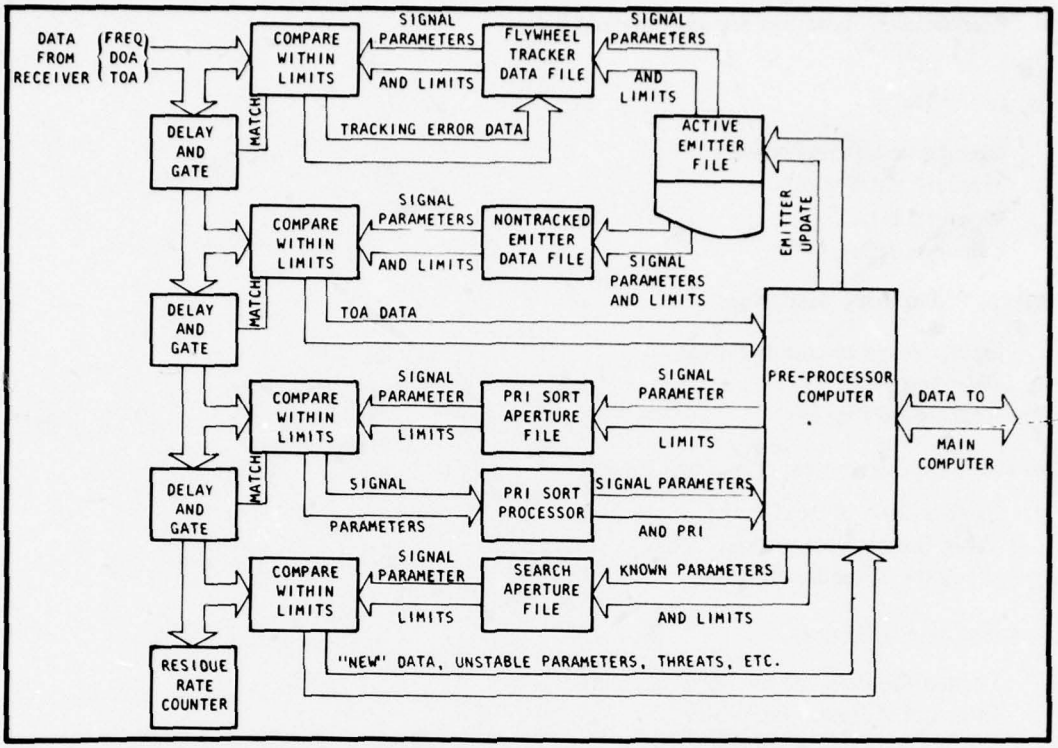


Fig. II-4—Signal sorter and associated circuitry

the emitter, eliminating the need to reprocess the emitter each time it is acquired. The matched condition deletes that emitter's pulses from the data stream. The remaining data stream consists of "new" and unmatched signals.

The third comparison serves to presort the data stream into the PRI processor to simplify the PRI sorting task. The preprocessor computer loads the PRI sort aperture file with values that allow the efficient sorting of the PRIs.

The final comparison is to priority search the data residue for special signals and to control the admission into the computer of unstable or unusual signals that could not be properly characterized in the preprocessor.

2. Flywheel Tracker

The flywheel tracker [1,2,3] keeps a running estimate of the signal parameters of the tracked emitter and determines the comparison limits to be used for the signal sorter, depending on the variability of the incoming data stream. The flywheel function compares the expected TOA of the next signal pulse with a real-time clock and generates a missing pulse flag if the clock time exceeds the expected TOA plus the TOA limit. It then adds the PRI to the TOA to generate the expected TOA of the next pulse from the given emitter.

The tracking function behaves as a predictor-corrector-type of parameter estimator. The predictor algorithm predicts the incoming signal parameters from the known characteristics of the emitter. Thus the DOA, for example, could be predicted from the known latitude and longitude of the emitter. The corrector algorithm compares the predicted signal parameters with the measured parameters and produces an error value, which is used to correct the emitter characteristics model. Thus the difference in the predicted and measured DOA, for example, is used to correct the ground location of the emitter. The equations for the predictor-corrector function are given as

$$\frac{\hat{S}_{pk}}{\hat{\sigma}_{pk}} = \phi(C_k) \tag{II-1}$$

$$\hat{C}_{k+1} = \hat{C}_k + \Psi(\hat{S}_{mk} - \hat{S}_{pk},\ \hat{\sigma}_{pk}) \tag{II-2}$$

where

$\hat{S}_{pk}$ is the kth predicted signal vector

$\hat{\sigma}_{pk}$ is the kth calculated limit interval vector

$\hat{S}_{mk}$ is the kth measured signal vector

C_k is the kth emitter characteristics vector

ϕ is a function relating signal parameters to emitter characteristics

Ψ is a function relating emitter characteristic errors to signal parameter errors.

Substantial work has been done in the development of adaptive tracking algorithms. When the dynamics of the emitter are well known, the tracking algorithms can get very sophisticated, even employing adaptive nonlinear Kalman techniques. The convergence rates of the estimates in this case can be very fast and bounded. The comparison limits can be adaptively adjusted, in response to changes in the signal environment. Even jittered and staggered PRIs can be efficiently accommodated.

Unfortunately, these adaptive algorithms are very complex, so the required calculations cannot be done by software unless the tracking information is updated occasionally, or unless only a very few emitters are to be tracked. The TOA estimates must be updated continuously, but this informaiton can be readily derived from the previous TOA measurement and the PRI estimate (which can be updated occasionally). The other emitter parameters do not generally change very rapidly, so that software adaptive tracking with periodic updates can be used; however, this technique does not use the adaptive algorithms to their maximum advantage. The adaptive calculations can be updated on a pulse-by-pulse basis by using dedicated hardware logic, but at a severe penalty in weight, power drain, cost and flexibility.

3. PRI Processing and Deinterleaving

The function of the PRI processor is to sort through the data stream and to determine the PRI of every emitter not listed in the active emitter file, or whose characteristics have changed enough to be outside the limits of the parameter prediction windows. [2, 4]

The incoming data stream, consisting only of pulses from emitters that have not been previously identified, is first passed through a presorting aperture (part of the signal sorter) to limit the number of pulses that must be processed at any one time. The aperture parameters and limits are dynamically determined by the preprocessor computer, based on the density of pulse inputs to the residue rate counter and the "new data" input to the computer. The presorting aperture must be large enough that so all signals from a single emitter will pass through, but not so large that a large number of different emitters are admitted, thereby substantially degrading the TOA sorting efficency. The incoming pulses are stored in a PRI buffer.

The PRI calculation is done in a straightforward manner, based on the TOA differences of data pairs in the PRI buffer. These differences form a tentative PRI value used to predict the TOA of the next pulse. The buffer is searched to determine if a pulse falls within the next TOA window. After a second or third match, the PRI is confirmed and all the matching pulses are removed from the data file. The process is then repeated to find the PRI from the other emitters.

A typical PRI deinterleaving program contains about 500 to 600 assembly language instructions. The calculations require a large amount of repetitive processing, so a processor load of 5000 program steps is not unreasonable to characterize the pulses from one emitter.

A high-speed computer performing the PRI processing should be able to characterize 50 to 200 emitters per second, depending on the computer and the processing algorithms used. This great a density of "new" signals is rarely found in a realistic environment (except at initial turn-on), so the PRI deinterleaving does not require a dedicated processor, but can be relegated as an additional chore for the preprocessor computer (or the main computer if it is not heavily loaded).

Processing PRI staggered and jittered pulses should not produce any difficulty. PRI staggered pulses appear to the deinterleaver as multiple emitters, all having identical signal parameters, including identical PRIs. These can be merged by the computer to produce one emitter with a staggered PRI. PRI-jittered pulses are left in the PRI buffer as a residue. When no more matches are found, this residue is processed with increasing TOA limits until the average PRI and PRI deviation are found. The residue in the PRI buffer can also be analyzed for linear PRI slides by taking pulse triplets, predicting values for the PRI and the rate of change of PRI, and looking for pulses that match the prediction windows.

As soon as the PRI of an emitter is determined, the emitter parameters are sent to the active emitter file and the flywheel tracker, so that the pulses from that emitter can be deleted from the incoming pulse stream as soon as possible.

4. DMA Controller

The output of the preprocessor consists of digital data containing the frequency, DOA, and the pulse width of each emitter in the signal environment, along with additional information for special and agile emitters. These data come out of the preprocessor in the preprocessor's internal data format—each data parameter is usually assigned its own data word. Sometimes these data words may have different word lengths tailored to each individual parameter. The multiplexer assembles these individual data parameter words into computer words in

a format suitable for storage and processing in the main computer. The data are then stored in the output storage buffer to await loading into the main computer [5, 6].

Since it is not possible to read and write the same section of memory at the same time, a DMA controller is used to synchronize the loading of the data into the main computer in such a way that the loading takes place only during that portion of the main computer's operating cycle when the memory is not being accessed by the main computer. In this manner, the loading can take place without disturbing the operation of the main computer. The DMA controller addresses the main memory directly, so that the data transfer can take place at the greatest possible speed.

D. Main Processor

The functions of the main processor are to control the operation of the ESM System; to identify the emitters and their platforms; to counter threats by taking appropriate action; to characterize and identify exotic emitters; to determine the geographic location of the emitters; to update the emitter files; to interface with the operators; to monitor system performance; and to perform auxiliary services, such as navigation calculations. A block diagram of the main processor is shown in Fig. II-5. The functions to be performed by the main processor are summarized below.

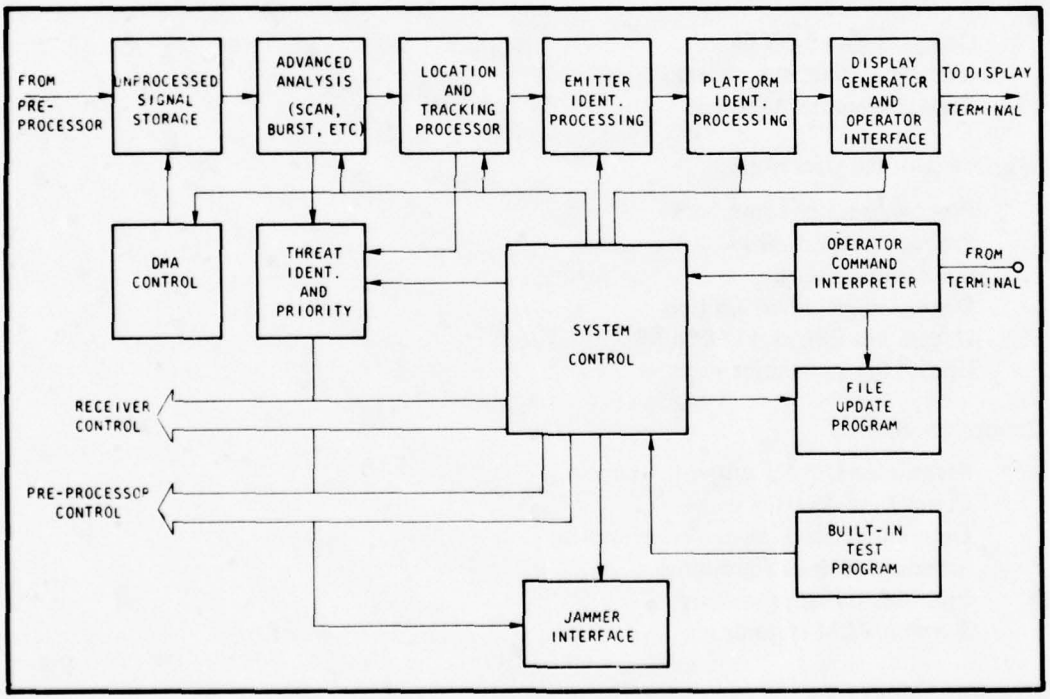


Fig. II-5—Main processor

Perform Signal Analysis

- Analyze scan types
- Analyze antenna patterns
- Analyze modulation characteristics
- Determine the emitter location

Analyze Exotic Signals

- Unstable
- Burst
- Chirp
- Pseudorandom

Identify the Emitters

- Compare the Emitter Parameter List (EPL) files
- Compare with Electromagnetic Order of Battle (EOB) files
- Maintain active emitter files
- Prioritize threats and take action

Identify the Platforms

- Compare with EOB files
- Identify associated emitters
- Correlate emitter locations

Interface with the Operators

- Process operator commands
- Produce video displays
- Sound threat alarm
- Display unidentified emitters
- Update the EPL and EOB files
- Update the environment plots

Control the System

- Sychronize DMA read/write operations
- Control the receiver scans
- Determine signal depopulation strategy
- Schedule analysis algorithms
- Prioritize threat processing
- Turn on ECM systems

Diagnose Failures

- Run built-in test programs
- Bypass defective systems

1. Operating System Software

The operation of the ESM processing system is controlled by a set of software programs that are known as the Operating System. The quality and convenience of the results produced by the computer depend entirely on the quality and thoroughness of the Operating System. The operating programs may be broken down by function into the following categories:

a. The Executive Program is responsible for managing and coordinating system resources in performing the various processing functions. It provides for the scheduling and dispatching of the various processing tasks on a priority basis and for the interpretation and handling of program interrupts. It allocates core and/or auxiliary memory storage and retrieves data from auxiliary memory.

b. The Processing Control Program controls the signal flow in the processing and emitter identification and reporting functions. It determines the additional processing required to identify a difficult emitter, determines the strategy to be used to depopulate the incoming signal data stream when the signal environment becomes too dense, and provides look-through timing for the processing system when an onboard jammer is active.

c. The Emitter Identification and Evaluation Program compares the parameters of the active emitters to parameters within a prestored emitter parameter file to identify the received emitters. It provides a tentative evaluation of the threat posed by signal sources that are not in the library or that may be identified with more than one type of emitter, and updates the emitter parameter fields to match the current environment. [1]

d. Operator Interface Program provides the operators with the capability of exercising control over the system and entering data into the system. It provides the operators with prompting and control menus to enable the operators to communicate their commands with minimum difficulty, presents digital data to the operator in a meaningful and understandable form, and displays the tactical situation to the operator, along with a geographic map of the area displaying fixes, tracks, and identified emitters.

e. Built-in Test Program monitors the performance of the ECM System during the mission to confirm the operational readiness of the system. It traces malfunctions to the replaceable unit level and determines whether processing can be continued in a degraded mode.

f. Miscellaneous Programs

Threat Processing and Self-Defense Control
Navigation Processing
Receiver Control
Extended Analysis Program
External Communications Control
Data Recordng and Control.

The threat analysis and self-defense programs and the executive interrupt handler are the only programs in the operating system that must be run in real time to rapidly accommodate these functions. The remainder of the operating system operates as time allows, according to a set of priorities stored in the executive program.

Typical program sizes for the operating system programs of ESM processors are

Program Type	Program Size
Executive	1500-4500
Processing Control	1000-5000
Emitter Identification	1200-1500
Operator Interface	300-4800
Built-in Test	600-1200
Miscellaneous	5000-6000
Program Core Size	9600-23000
Data Storage Area	8400-15000
Total Memory Size	18000-38000

These are illustrated in Fig. II-6. The sizes indicated in the table are not intended to serve as program size limits, but to show the typical size and variability of the sizes of the various programs. In general, the analysis programs do not show much variability in program size, since the amount of computation required to solve a given problem is determinate. The operating and control programs, and especially the operator interface programs, can be any size, depending on the scope of the problems to be solved, the amounts of automation desired in the system, and, to a large extent, on the amount of sophistication, subtlety, and operator convenience that the programmer wishes to design into the system. The programs listed are the ones normally stored in computer memory during operation. A large amount of additional programming is normally stored in auxiliary disc or drum storage to handle unusual situations when different techniques and special purpose programs are required.

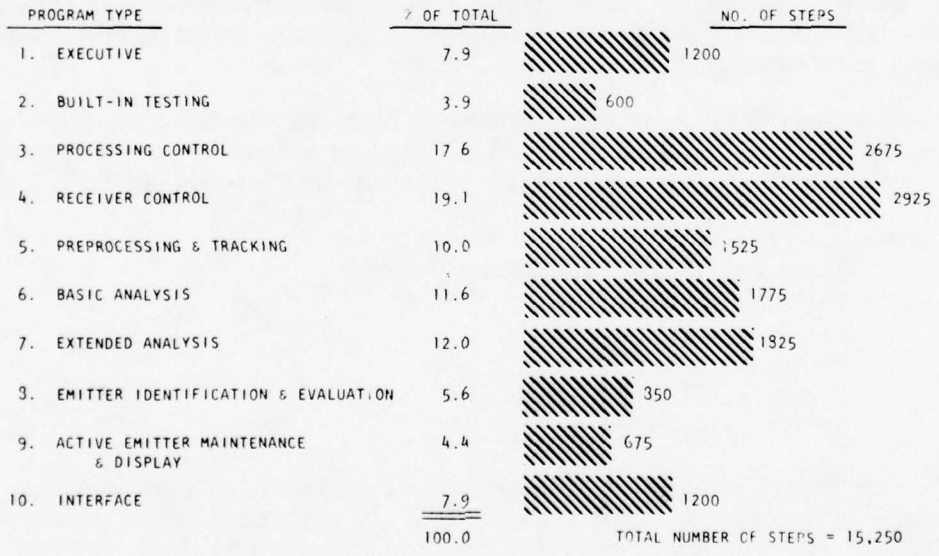


Fig. II-6—Typical program sizes

2. Emitter Identification and Evaluation

The emitter identification process consists of successive comparisons of measured emitter parameters with a succession of emitter parameter lists, in order to identify the most likely emitter to be generating the measured signal. Figure II-7 shows the major files used to perform the comparison. Signals intercepted by the receiver are automatically analyzed by the preprocessor to determine the signal parameters. When the analysis of the signals from any given emitter is completed, the emitter parameters are sent to the preprocessor's active emitter file for tracking and to the main computer for identification.

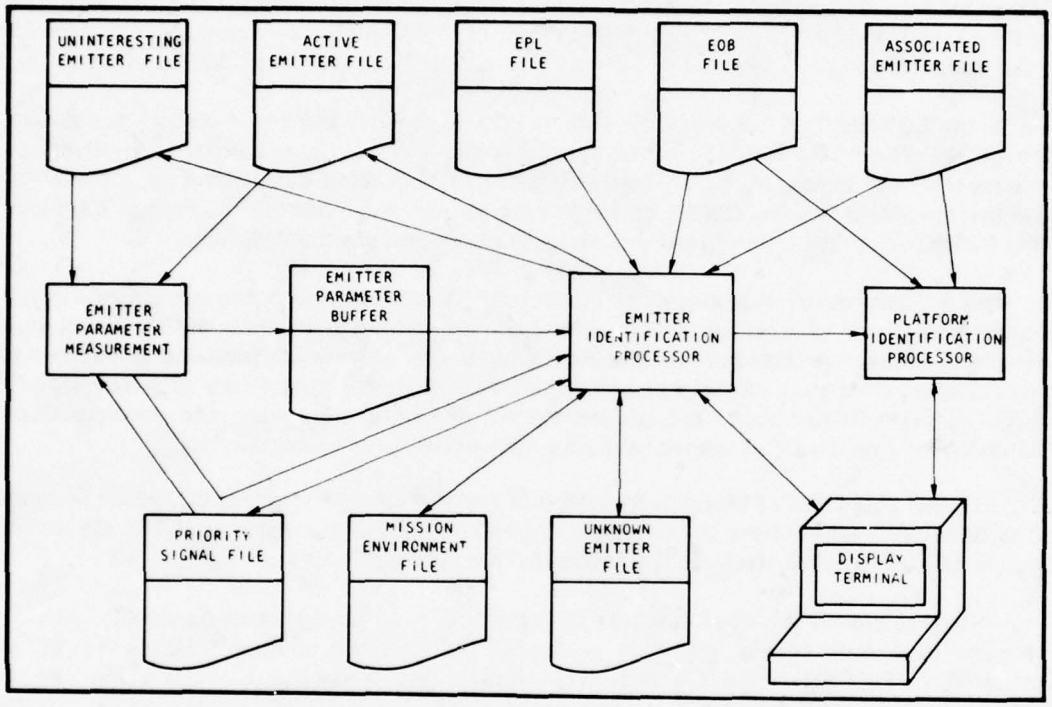


Fig. II-7—Data processing flow

If the analysis cannot be performed successfully (not all electromagnetic signals can be automatically classified), the signal parameters that have been identified are sent to the operators for manual computer-aided analysis or, in a fully automatic system, the residue signals are tagged as unidentified.

The emitter characterization parameters from the preprocessor are input to the main computer on a periodic basis under DMA control and are stored in an emitter parameter input buffer. Emitter parameters remain in this file until the emitter is identified by the emitter identification processor, in which case they are deleted from this file and are moved to the Mission Environment file. New emitter parameters in the input buffer are first compared with

entries in the Mission Environment file to determine whether the signal has been previously identified. If a match is found, the Mission Environment file is updated to reflect the new data and the parameters are deleted form the input buffer. If there is no match, a similar comparison is done with the parameters stored in the Unknown Emitter file, to assure that the emitter has not been previously processed. Successful correlation here results in updating the parameters in the Unknown Emitter file, and another attempt is made to process the updated information. The signal intercepts that do not match either file are considered new signals and are compared with an a priori data base consisting of the EOB file and the EPL.

The EOB file contains all available data concerning the emitters that have been recently active and are suspected to be active in the area. The file contains exact values for the emitter parameters and the tolerances for those parameter values for specific emitters. The data are largely derived from the Mission Environment files of intelligence-gathering missions. A match of the emitter parameters with the parameters of the EOB file results in identifying the emitter.

If the EOB file does not produce an identification, an effort is made to identify the emitter through the EPL. The EPL file contains a list of the generic parameter data on all known emitters that may appear in the environment. The tolerances on the parameters are often so wide that several emitters in the file may match the emitter parameters in the emitter parameter input buffer. As a result, more than one match is often found in the EPL file.

Emitter parameters that cannot be immediately associated with one given emitter, either because there are no matches or because there are multiple matches within the emitter identification files, are stored in an unknown emitter file, and the preprocessor is directed to gather additional data on the emitter to facilitate further processing. If the advanced analysis processing still fails to identify the emitter, the list of emitter parameters and the most likely candidates for a match are presented to the human operators for a decision.

Successfully identified emitters are stored in the Mission Environment file, which contains all pertinent data on emitters intercepted and identified during the mission, and in the Active Emitter file, which keeps track of all emitters that are currently active.

After the emitter has been successfully identified, an attempt is made to identify the platform and to identify emitter groups by comparing the identified emitters with the Associated Emitter file. This file contains a matrix of emitters and platforms, identifying the emitter configuration of a given platform. It is possible to identify the platform containing the emitter by comparing several colocated emitters with the Associate Emitter file. Once the platform has been identified, other emitters that are expected to be on the platform can be identified. This process is especially useful in identifying emitters that have produced several matches in the EPL file.

The Associated Emitter file also contains matrixes of emitters showing the emissions generated during the various modes of operation of a given emitter system. By using the Associated Emitter file, future signal intercepts can be predicted, resulting in a significant enhancement of system response speed.

As shown in the list on page 16, the Emitter Identification and Evaluation Program size is not very large, typically requiring only 1200 to 1500 instructions. The program flow is also

very simple, consisting mainly of searching a series of successive data files and comparing the input parameter file to the stored data parameters and parameter limits in an effort to produce a match.

From a system specification standpoint, the main problem with the emitter identification process is the sheer size of the data library that has to be searched. Although a list of the most urgent threats could be stored in as few as 500 storage locations, a more realistic list of the major emitters to be found in a given area would take 3000 storage words. Worldwide emitter tables would require well over 30,000 storage words for the EPLs alone.

Another problem is the time required to make all the comparisons necessary to identify the emitter. Most high-speed processors are able to identify less than 10 emitters per second on a continuous basis (although they can acquire and track about 100 emitters per second in the preprocessor). The identification speed depends on the speed of the processor and on the speed of data retrieval from the storage medium.

The system designer must decide how to allocate the EPL data among the storage media. Core storage is very fast, but expensive. Magnetic tape storage is very slow and inexpensive. Drum or disc storage is intermediate in both speed and cost. A good compromise would be to place all threat emitter data in core for fast retrieval. Area nonthreat parameter lists can be stored on disc or drum. All other nonthreat emitters can be stored on magnetic tape for later analysis.

Data retrieval from magnetic tape is usually so slow that the overall emitter identification throughput is independent of the speed of the processor. For modern high-speed processors, the same is true for disc or drum storage. For data stored in core, a faster processor can naturally make the required comparisons faster, and, therefore, is able to identify more emitters per second. The designer must, therefore, be able to identify these processor characteristics that influence processing throughput.

III. ARCHITECTURE

An attempt to describe computer/processor architecture is complicated by the many facets these machines exhibit to the many fields of specialization that are necessary to their design and function. The hardware engineer may view computers by the technology involved; LSI, CMOS, bipolar, core, pipelining, DMA, data bus, word size, etc., are all terms that may describe one processor as opposed to another. The software engineer may describe computer architecture in yet another manner. The term *architecture* is used also to describe the attributes of a system as seen by the programmer; i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation. This definition of architecture specifically excludes details of hardware implementation. The instructions and registers that programmers "see" are part of the architecture, but the data buses are not. For example, the IBM 360/30 uses 8-bit data paths, the 360/40 uses 16-bit paths, and the 360/50 uses 32-bit paths, but all three are the same architecture, and can execute the same programs. As another example, the PDP-11 Unibus is not an integral part of the architecture (indeed PDP-11s have been built with at least three different bus structures), but the use of dedicated memory locations for communication with I/O devices is an architectural feature because the programmer does not see the unibus, but he does see the dedicated I/O registers. [7]

In other words, there is no concise simplification of computer architecture in spite of the extensive use of this term. Yet we cannot do without it. An attempt will be made in this section to clarify for the reader what architecture means both to the hardware engineer and the software programmer. The bulk of computer processing improvements over the last two decades have risen out of technology advances rather than out of architectural changes, and this principle is likely to remain in effect for at least another generation or more of computer systems. It is more promising, then, for the Navy to adopt an already successfully demonstrated extant computer architecture, commercial or military, and to use that architecture to reap the benefits of technological advances while enjoying the benefits of software stability. The selection of an existing architecture carries with it an understanding of the strengths and weaknesses of that architecture and also a useful inventory of support and applications software already developed. [8]

A. Hardware

From the simple "hobby" computers to the largest "number crunchers," the basic concepts behind all computer systems are the same. A conceptual computer with the basic functional modules to be found in all computers is shown in Fig. III-1. All computers consist of a memory module for storing data and information and a central processing unit (CPU) for doing the processing. The CPU contains either an accumulator or a general register file for storing data, an arithmetic and logic unit (ALU) for processing the data, a processor control unit, an instruction register to store instructions, a program counter, and a memory address register. [6,9]

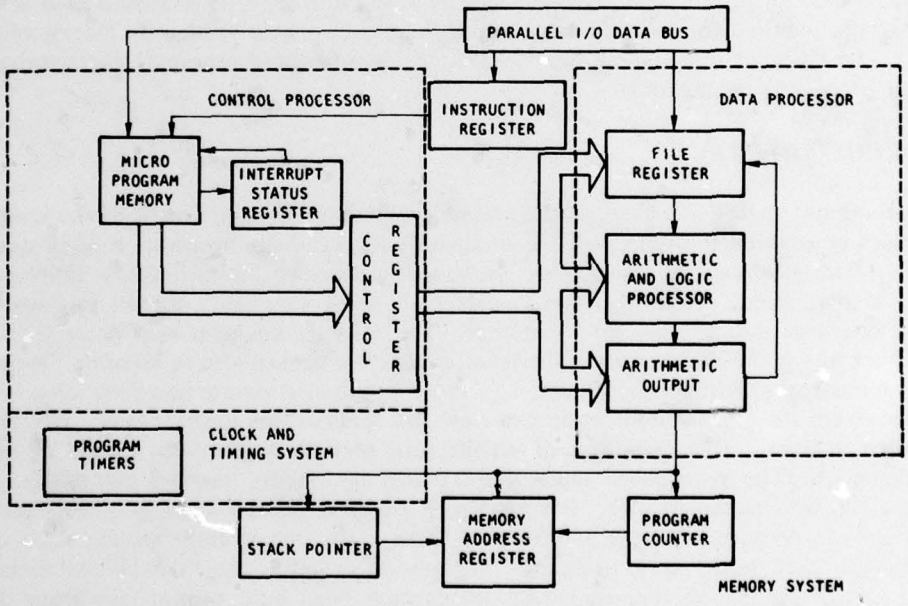


Fig. III-1—Digital input/output data

The basic operation of any computer is as follows:

1. The program counter contains the memory location of the next computer instruction, which is stored in the main memory. Operation begins when the contents of the program counter are loaded into the memory address register and sent out via the memory address bus.
2. The program counter is incremented to point to the memory location of the next following instruction.
3. The instruction code is read from the main memory via the data bus and is stored in the instruction register. The contents of the instruction register address the microprogram memory.
4. The control word in the microprogram memory (at the address location corresponding to the instruction) is sent to the data processor via the control register. The different bits in the control word select the source register in the general register file, the function to be performed by the ALU, and the disposition of the result of the operation.
5. Sometimes the microprogram memory is clocked through more than one address for each instruction in the instruction register to generate more than one microinstruction for each machine instruction. The sequence of microinstruction addresses used can be varied, depending on the contents of the status register, thereby allowing conditional program branching.

The above sequence describes a register-to-register operation. In general, the microinstruction can load any register from any other register through the ALU and the output switch. Therefore, the address register can be loaded by the microprogram from any other register or from the data bus, and data may be input or output via the data bus. The microprogram can also load the program counter and thereby cause unconditional branching to another program segment or to a subroutine.

This explanation of the way computers operate may be extremely elementary, barely scratching the surface of the operational complexity of a modern computer system. Nevertheless, the presentation of basic operating concepts should be sufficient to understand some of the architectural features that can be used to increase a computer's processing throughput.

In most modern computers, the CPU is usually composed of an integrated circuit (IC) microprocessor (sometimes composed of a number of IC modules) and associated circuit modules. Microprocessor technology is now in a rapid state of flux, with new microprocessors continually being introduced by the various manufacturers. The problem, then, is how to evaluate the ESM processor with respect to each new device in order to determine which of them would perform the ESM processing tasks at the greatest speed, with the greatest throughput.

1. Instruction Timing

The basic computer is composed of a CPU memory section (plus an I/O interface to communicate with the outside world). Therefore, the processing speed of the computer depends on the processing speed of the central processor and the access time of the computer memory.

To determine how these factors contribute to the total instruction timing requires an analysis of the basic instruction execution sequence of a computer (shown in Fig. III-2). The sequence may be divided into two main parts: the instruction fetch cycle, and the instruction

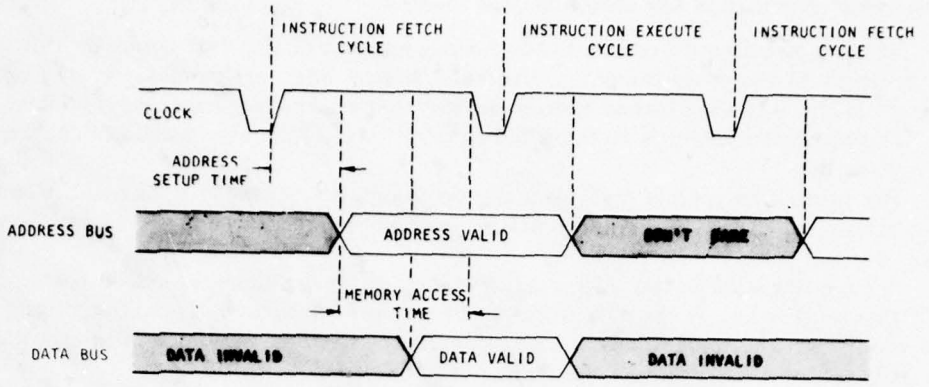


Fig. III-2—Microcomputer instruction execution sequence

execute cycle. For the instruction fetch cycle, the CPU logic outputs the contents of the program counter register to the memory address register at the positive edge of the clock. A short time later, the address appears on the system address bus. While the external logic is responding to the address change, the CPU increments the program counter by one, thereby pointing to the next instruction to be fetched. The processor now waits for the memory output on the data bus to become stable. After a sufficient time (called the access time of the memory) has elapsed, the clock goes low and the instruction on the data bus is stored in the instruction register. Once the instruction code is in the instruction register, it triggers a sequence of events that constitutes an instruction execution cycle. It is quickly decoded by the microprogram control unit into one or more control words (also called microinstructions). These control words are sent to the CPU to control the processing. A portion of the control word is also fed back to the microprogram control unit to control the sequence of microinstructions. [6]

2. Pipelining

In a pipelined architecture, Instruction 2 is fetched immediately after Instruction 1 (Fig. III-3). Thus, while the memory is accessing Instruction 2, the central processor can be executing Instruction 1. In this way, the memory utilization rate and average instruction speed are nearly doubled for programs that consist of long instruction sequences where all the instructions invariably follow one another [6,10—12].

When a pipelined computer has to execute a program branch (as a result of a compare and branch instruction, for example), the pipeline circuitry will have already fetched the next instruction of the regular sequence (or in the case of an unconditional branch, the contents of the next higher location in the program memory). The instruction register will contain the wrong instruction and must be cleared, any steps taken to begin the instruction must be reset, and the fetch cycle for the correct instruction must be initiated. As a result, a pipelined computer will usually execute branch instructions more slowly than a nonpipelined computer would.

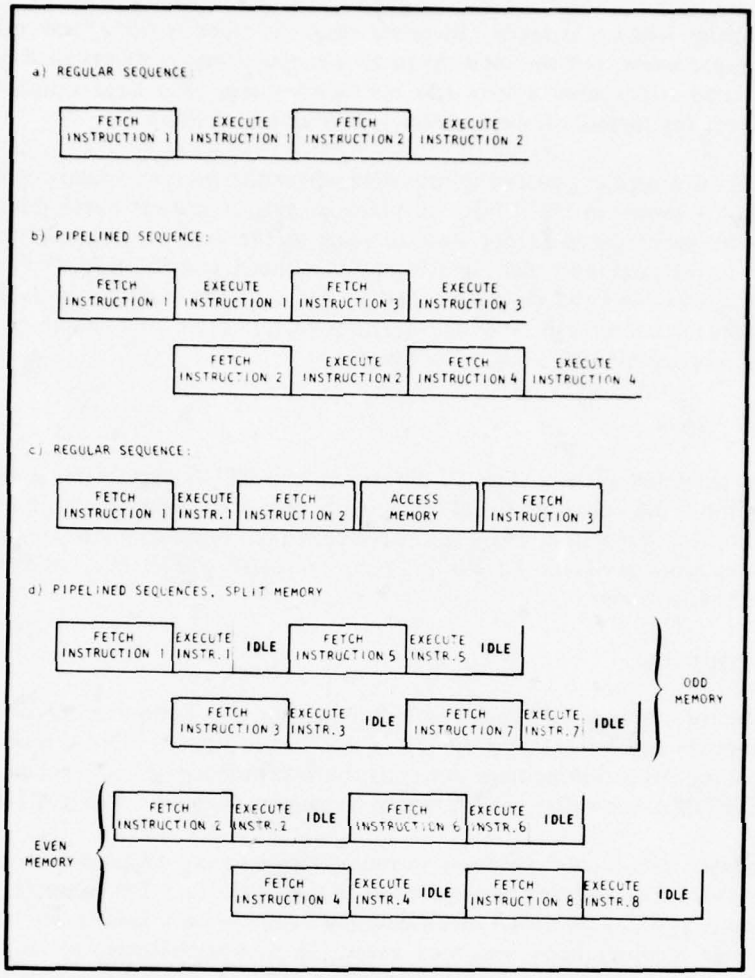


Fig. III-3—Regular vs pipelined instruction sequency timing

3. *Effect of Memory Speed*

It is apparent that for processors operated by a constant-frequency clock, the minimum clock period of the pipelined processor is determined by either the memory access time or the central processing unit's microinstruction cycle time, whichever is greater. Therefore, if the memory access time is shorter than the CPU's basic cycle time (less the address setup time), the operating throughput of a computer is determined almost entirely by the basic cycle time of the central processor. On the other hand, if the memory access time is a bit larger than the CPU's basic cycle time, the operating throughput is determined by the memory access time (plus the address setup time). [13]

To operate with even slower memories, the system designer has two options — either take more than one clock cycle to access the memory, or stop the clock entirely and have the

memory acknowledge when it is ready. In either case, the clock is run at the minimum cycle time of the microprocessor, but the total instruction execute time is determined largely by the memory access time, since only a very few instructions require a large number of execute cycles without accessing memory for more data or further instructions.

An example of a regular processing sequence where the processor takes two clock cycles to access memory is shown in Fig. III-3c. In practice, most processor clocks do not run asynchronously, so the processor does not start as soon as the memory acknowledges that it is ready, but waits to the next clock tick (which may be either a quarter, half, or full clock cycle, depending on the CPU and the clocking scheme used). If the memory is so slow that the memory access time is more than twice the microinstruction cycle time, some of the lost processing time can be recovered by using a "split memory."

4. Split-Memory System

In a fully pipelined system (Fig. III-3b) where the CPU's cycle time is faster than the memory access time, the operating speed of the computer is determined almost entirely by the memory access time. The system clock can either be slowed down to match the memory access time (or a slower processor could be used), or the processor will sit idle, waiting for the next instruction fetch cycle to end.

5. Direct Memory Access

Direct Memory Access (DMA) is a technique for the bulk transfer of memory data between a computer and an external device or another computer. DMA transfer is directly related to pipelining, with the memory access at the alternate leg of the pipeline being directly controlled by the DMA controller. DMA timing sequences are shown in Fig. III-4.

In a computer with a split memory, various techniques can be used to implement DMA transfer without seriously affecting the operation of the computer. For example, in a nonpipelined split memory system, the DMA can access the even memory bank while the computer is accessing the odd memory bank and vice versa. In a pipelined split-memory system, the memory can be subdivided further into blocks. The DMA controller can access one block of memory while the computer is operating on another memory block, with no interference between the two operations. Although DMA operations do not increase the operating speed of the computer itself, they can provide a significant increase in the ESM System operating speed, since in most ESM systems, large amounts of data must be moved back and forth among the various processors.

6. Microprogrammability

A microprogrammable computer allows the users to insert their own control and sequencing microinstructions into the microprogram memory of the CPU (see Fig. III-1). This gives the computer a set of unique user-defined instructions for performing special, job-related repetitive or time-consuming procedures. [12, 14]

The major advantage of having the special instruction sets is that these procedures can be executed without continually referring back to the main memory for instructions, thereby saving considerable processing time. A ten-step procedure, for example, which would normally

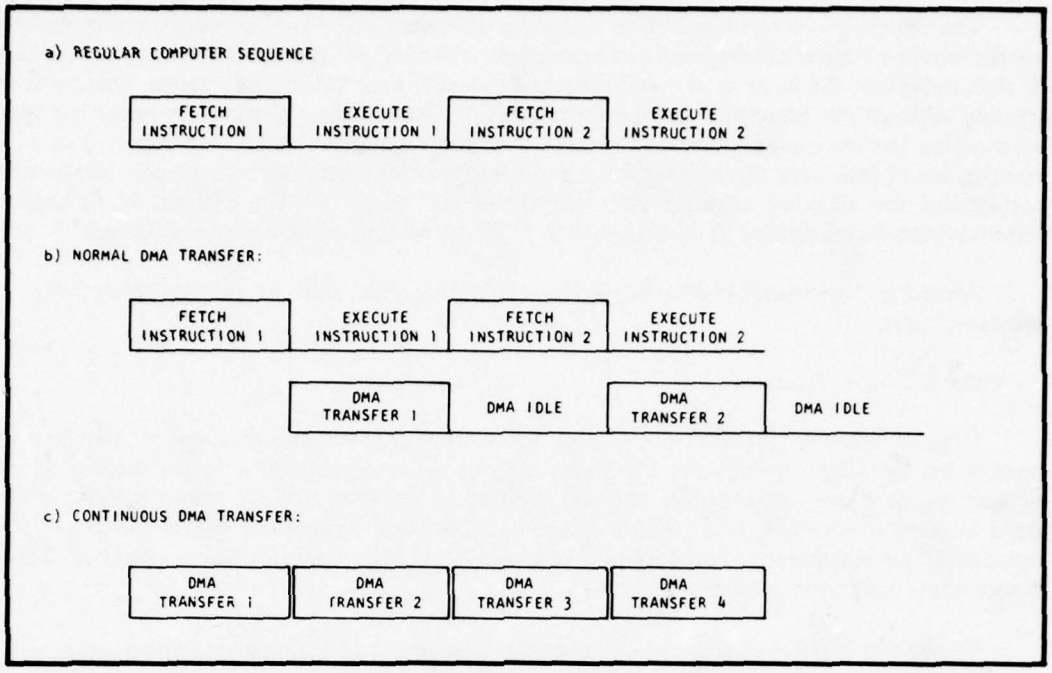


Fig. III-4—DMA transfer timing

require ten instruction fetch cycles to implement, can be carried out after a single instruction fetch cycle. In this case, the time saved equals nine memory access times.

In an ESM processing system, the most likely candidates for microprogramming are the following procedures:

1. The adaptive tracking of the parameter values and the search limits update procedures

2. The procedure to calculate the next TOA

3. A compare-between-limits procedure for PRI searching and for emitter identification searching

4. The PRI procedure for comparing pairs of TOA values and generating the next TOA window

5. A masked comparison procedure for comparing packed data

6. Multiple load and store procedures for moving blocks of data between internal registers and memory

7. Special table-search procedures, such as search every *N*th byte or every *N*th word, to facilitate the searching of numerically ordered parameter lists.

The increase in overall computer operating efficiency due to microprogramming depends on the length of the microprogrammed procedures, the ratio of internal-to-memory access steps in the procedure, the ratio of the microinstruction cycle time to memory access time, and the fraction of time the procedure is used during normal processing. The gain in operating speed when using the microprogrammed procedure over the regular procedure is typically 3 to 1. A comparison of processor throughput for a microprogrammed computer vs a regular computer in performing the adaptive tracking task was shown on page 14. The processing throughput improvement varied from 2 to 1 through 4 to 1 for the various processors investigated.

Formulae for the increase of processing throughput due to microprogramming are presented later.

7. General Purpose Registers

General purpose registers are useful for storing intermediate results of mathematical operations, as index registers for modifying addresses for repetitive (looping) calculations, as pointer registers (page pointer) for addressing different blocks of data, for block transfer of data from location to location, or as multiple accumulators for arithmetic or logical functions. The number of general purpose registers in a processor is usually chosen to be a power of 2 from binary addressing considerations. [15, 16]

Register-to-register operations are normally performed in a single microinstruction cycle, so a large number of general purpose registers can provide for extremely rapid data processing by decreasing the number of external memory references required. Register-to-register instructions are usually constructed in single-word format, which decreases program storage space and also increases the processing speed.

General purpose registers are sometimes arranged in two banks, with each bank independently available for data storage. Programs that are called repeatedly into operation and applications that require rapid switching from one task to another can each be assigned a register set for its own use. This increases processing speed by eliminating the need to change the contents of the general registers each time a task is changed.

8. Processor Word Length

In order for the computer to operate at a high speed, it is important that the computer word length not be too small. The time required to fetch a word from memory is essentially independent of the word length. Fetching a two-word instruction, however, takes about twice as long as fetching a one-word instruction. Therefore, from the point of view of the instruction set, a computer word must be sufficiently long to allow all but a few of the instructions to be specified in single-instruction words. On the other hand, instruction word bits that are not used are wasted, so that overly long instruction words simply increase the cost without increasing the performance.

The cost-vs-performance tradeoff has been studied by various minicomputer manufacturers, with the result that the majority of manufacturers have adopted the 16-bit computer word size as a standard. These 16-bit computers can perform all the fast register-to-register operations with one-word instructions, although most computer operations that address the main

memory directly require two-word (32-bit) instructions. Special addressing techniques, such as indexed addressing and memory paging, permit some main memory operations with one-word instructions. It may be considered that a 16-bit processor word is a reasonable minimum word size for the ESM processor instruction words, whereas a 32-bit word size is perferred if the processing throughput rate must be a minimum.

These same arguments hold for data words, except that for some mathematical operations, such as shifts, data stored in two short memory words takes *more than twice as long* to process as it would take to process the same data if it were stored in one long memory word. This is one reason why large number-crunching computers tend to have 48- to 64-bit memory words. On the other hand, data words that are longer than required waste memory.

It is possible to increase the data storage efficiency by packing more than one data word into each of the extralong memory words. However, for very short data words, a significant portion of the computing time gets spent in packing and unpacking the data bits, and the advantages of using these extra long memory words are lost.

The optimum data word size for the ESM System can be determined from an analysis of the amount of information required to uniquely locate and identify each emitter of interest in the environment. The actual number of required data bits depends on the proposed data analysis algorithm, which, in turn, depends on the specific application envisioned for the ESM System.

An estimate of the number of bits that should suffice to represent each of the various signal parameters is presented in Table III-1.

In selecting the processor data word size, an attempt should be made to avoid splitting up any of the parameter data words into two processor words. Each portion of the split word must be processed separately, and then the separately processed portions must be logically connected together with further processing; thus a data word split in two takes more than twice as long to

Table III-1 — Representative ESM Parameter Word Size Requirements

Parameter	Range	Increments	Word Size (bits)
Angle of Arrival	0—360°	0.35°	10
Time of Arrival	0—200	0.2 μs	20
Signal Amplitude	-120 to +8 dBm	1.0 dBm	7
Pulse Width	0.25 μs		
	0—12.5 μs	0.1 μs	8
Frequency	0.5 GHz	5 MHz	
	5—10 GHz	10 MHz	11
	10—15 GHz	20 MHz	
or			
Frequency	2—4 GHz	1 MHz	
Total			56 bits

process as a single data word. For example, in Table III-1 the minimum processor data word size is 20 bits to accommodate the 20-bit TOA word. All data for one emitter pulse could be stored in three 20-bit words, two 28-bit words, or one 56-bit word.

In most general purpose computers, the processor instruction word size and the data word size are the same. This allows each general purpose memory location to be used to store either data or instructions, whichever a given program may require. Since a large variety of programs may be run on a general purpose computer, this arrangement results in the most efficient utilization of the main memory.

In a special purpose ESM processor dedicated to performing only one given task, the instruction memory and the data memory can be kept separated, with each type of memory having a different word length suited to the given application.

9. *Hardware Options*

To enhance programming versatility and to increase computation throughput, some processors utilized separate IC hardware to perform special functions that would take very long to perform with software [15—17]. The hardware often includes the following functions:

1. Floating-point arithmetic hardware, which performs precision floating-point instructions much faster than ordinary floating-point software subroutines

2. Double-precision multiply and divide for direct processing of double-length operands

3. A trigonometric function package, which includes sin, cos, arctan, and other trigonometric functions.

In ordinary operation, most ESM processors do not make extensive use of floating-point or double-precision arithmetic, so these functions need not be included in an ESM processor as special hardware. The trigonometric function package could be useful for navigation and for emitter location algorithms if these calculations are performed fairly often. If the calculations are done infrequently, they may as well be implemented in software.

Some newer ESM Systems are turning to sophisticated mathematical techniques, such as adaptive Kalman filtering, to perform the adaptive tracking algorithms and fast Fourier transforms to do the PRI processing. These techniques normally take too much processing to allow a high processor throughput if they are implemented in software. They may prove to be practical, however, if performed by a special function hardware addition to the computer.

10. *Multiprocessing*

The system designer must decide how to allocate the computer resources that will do the processing. The obvious first approach would be to use a big general purpose computer to do all processing. In this case, all processing would be done in one central processor, and all data would be stored in one central memory, as shown in Fig. III-5a.

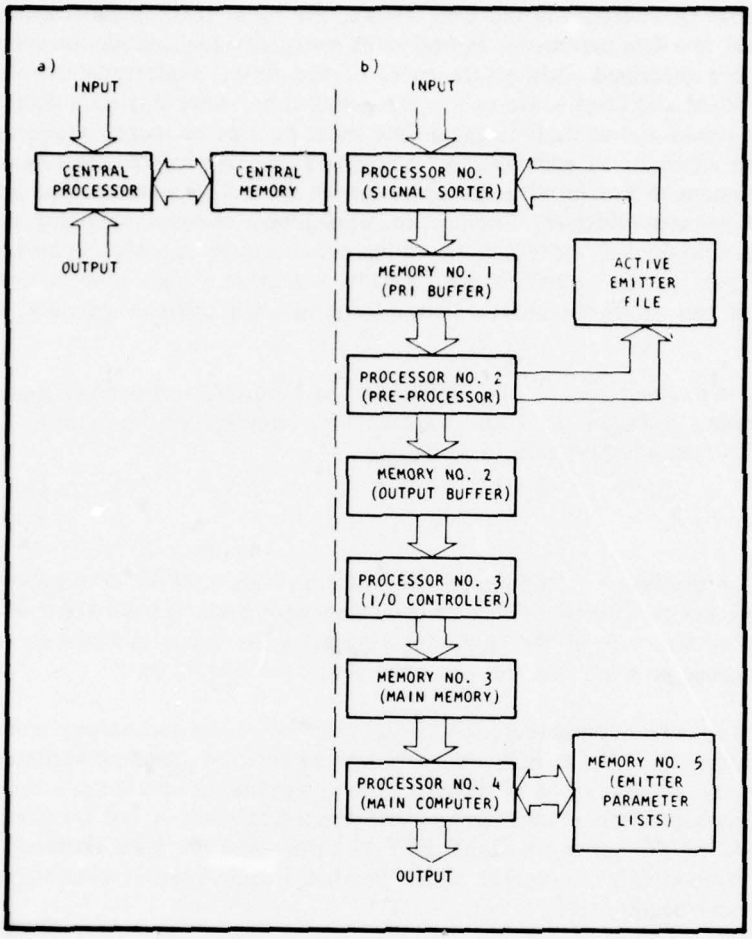


Fig. III-5—Possible processor configurations

As is pointed out in Section IV, the entire processing capability of an AN/UYK-20 is not sufficient to track more than ten emitters on a pulse-to-pulse basis, so it becomes apparent that at least some processing will have to be done by hardware, or special purpose software processors.

The approach taken by most modern ESM Systems is to divide the processing task among as many processors as possible. Since each processor works on a small segment of the problem and all processors operate simultaneously, the overall system throughput can be increased considerably by using multiprocessing. Task size and processor speed are selected so that each processor works at about the same rate, enabling the data to flow smoothly through the system with no backup.

The system illustrated in Fig. III-5b is such a system. The approach shown, where no more than two processors access each data memory, can be designed with minimal interference

between processors by interleaving the data transfer timing of the two processors. Since each processor accesses two data memories, as well as its own individual instruction set memory, the processors can use pipelined architecture most of the time. Unfortunately, this technique wastes memory space and loses some of the processing time, since duplicate data must exist in several memory blocks and some processing time must be used to merely move data from one memory block to another. In addition, this involves a complex interconnect and highly complex operating system to synchronize the operation of the entire system. The architecture of such a system is not cost-effective. The cost of developing a complex operating system capable of synchronizing a number of processors operating simultaeously has been demonstrated over a long number of years to be overwhelming. Finally, it implies a high level of unreliability for the system, as it can never be proven that the system will operate correctly under all circumstances.

The above discussion viewed only one aspect of hardware architecture that is most pertinent to processing throughput. Other authors view architecture from more technological aspects, as chip composition, bit-slice, devices, etc.

B. Software

The joint Army/Navy Computer Family Architecture (CFA) committee has largely evaluated architecture on a software (higher level language) basis. They have defined computer architecture as "the structure of the computer a programmer needs to know in order to write any machine language program that will run correctly on the computer."

With a well-specified architecture, details of data bus width, technology (core memory vs semiconductor memory, TTL vs ECL circuits), implementation speed-up techniques, physical size of computer, etc., need not be of concern to the programmer and hence are not part of the architecture. This separation of architecture and implementation is not a radically new idea. The IBM System/360-370 series, the DEC PDP-11 series, and the Data General NOVA series are just three examples of computers in which this has already been successfully accomplished to a greater or lesser degree. [18]

Thus, software architecture operates relatively free of technology, permitting computer performance technology to advance without significantly affecting the software approach to programming the machine. For example, the PDP-11 using the same software architecture delivers significantly different throughput performance with hardware architectures of core and bipolar memories, see Table IV-9. Yet these machines are programmed in the same manner. On the other hand, the software performance of the AN/UYK-21 (PDP-11) and AN/UYK-20 are quite different even with the same technology (core memory). The ensuing discussion is derived from the extensive work of the CFA, which is also known as the Military Computer Family (MCF) program. The goal of this program is to provide quantitative standards of performance in support of their recommendations for the standardization of military computers/processors.

1. *Absolute Criteria*

The CFA selection commitee specified nine *absolute criteria* [19] that they felt a candidate computer architecture needs to satisfy if it is going to meet the requirements of future military

computer systems. All absolute criteria (with the exception of the subsetability criterion) had to be satisfied by an implementation of the architecture, which was operational by January 1, 1976. This eliminated speculative decisions based on promises or potential solutions that looked inviting, but might not come to fruition. Failure to satisfy any absolute criterion resulted in the eliminaton of the architecture from further consideration. The nine absolute criteria are given below. The formal statement of each criterion is underlined, while explanations and examples are not underlined. Many comments that follow the definition of an absolute criteria are the result of the experience gained when the CFA committee evaluated the nine candidate architectures against these criteria [20]. Table III-2 shows which absolute criteria each candidate architecture passed or failed.

a. Virtual memory support

The architecture must support a virtual-to-physical address translation mechanism. The intent of this criterion is to take advantage of the widely used feature of many machines known as virtual memory. Many advantages accrue to architectures that support virtual address translation mechanisms, the most notable of which is the ability to simplify programming by freeing the programmer of explicit management of his primary memory and providing a mechanism for keeping only the active portions of a program in high speed memory.

The answers for this criterion listed in Table III-2 are not controversial, except for the AN/UYK-20. This architecture provides the page registers necessary for relocation, but it does not limit the ability to change these registers to privileged programs. Some members of the CFA committee felt that preventing user access to the page registers was a necessary aspect of virtual memory; others disagreed. The full CFA committee voted to fail the AN/UYK-20 on this criterion. The ROLM-1664 and SEL-32 both failed this criterion because each of these architectures provides a mechanism commonly known as "bank switching," which the committee felt was not an adequate memory translation mechanism.

b. Protection

The architecture must have the capability to add new, experimental (i.e., not fully debugged) programs that may include I/O without endangering reliable operation of existing programs. The intent of this criterion is to provide a mechanism in the hardware for aiding software development and to keep certain catastrophic software failures from occurring in the field. Architectures that use a privileged mode to protect vital registers and system resources generally meet this criterion.

The AN/UYK-20 failed this criterion because it lacks memory protection; any user can modify the contents of the relocation registers and thereby read and write any word in memory. Another generic way for an architecture to fail the protection criterion is for a program to have the ability to put the machine into a noninterruptible state for an indefinite time. Architectures that permitted nonterminating instructions were carefully examined to determine whether these were, or were not, interruptible.

c. Floating-point support

The architecture must explicitly support one or more floating-point data types with at least one format yielding more than 10 decimal digits of significance in the mantissa. The significance measure was determined as representative of the most stringent requirements actually encountered.

Table III-2 — Candidate Architecture Value for Absolute Criteria

Absolute Criterion	Candidate Computer Architectures								
	IBM S/370	Inter-Data 8/32	Rolm AN/UYK-28	DEC PDP-11	Univac AN/UYK-7	SEL 32	Burroughs B6700	Univac AN/UYK-20	Litton AN/GYK-12
1 Virtual memory	Y	Y	N	Y	Y	N	Y	N	Y
2 Protection	Y	Y	Y	Y	Y	Y?	N	N	Y?
3 Floating point	Y	Y	Y	Y	N	Y	N	Y	N
4 Interrupts/traps	Y	?	Y	Y	Y	Y	Y	Y	Y
5 Subsetability	Y	Y	Y	Y	Y?	Y	Y?	Y	Y?
6 Multiprocessor	Y	Y	Y	Y	Y	Y	Y	Y	Y
7 I/O Controllability	Y	Y	Y	Y	Y	Y	Y	Y	Y
8 Extensibility	Y	Y	Y	Y	Y	Y	Y	Y	Y
9 Read-only Code	Y	Y	Y	Y	Y	Y	Y	Y	Y
SUMMARY	Y	?	N	Y	N	N	N	N	N

Y Yes, meets criterion.

N No, fails criterion.

Y? Yes (but with some reservations).

? Unresolved.

Note: Table adapted from Ref. 18.

The AN/GYK-12 failed this criterion because it does not support floating-point operations. The AN/UYK-7 failed because it supports a single, 64-bit, floating-point format with only 31 bits (9.2 decimal digits) of mantissa. Because this is so close to the borderline, one might reconsider requirements on significance to determine how firm the 10-decimal digit criterion is. (Had the AN/UYK-7 looked like an otherwise excellent architecture, it is likely that the committee would have relaxed the floating point absolute criterion for it.)

d. Interrupts and traps

It must be possible to write a trap handler that is capable of executing a procedure to respond to any trap condition and then resume operation of the program. For example, if the processor receives a page-fault trap from the address translation unit, it must be able to request that the appropriate page be brought in from secondary storage and then resume execution. If resumption of execution is logically impossible (e.g., an attempt to store an operand into a read-only segment of virtual memory or fetch an instruction with a parity error), then the trap handler should be able to abort the program with an indication of which instruction and/or operand caused the termination.

A similar requirement exists for interrupts: *The architecture must be defined such that it is capable of resuming execution following any interrupt* (e.g., power failure, disc read error, console halt).

Another intent of this criterion is to permit extensions and subsets of an architecture to operate correctly so that programs can be upward or downward compatible. The subsets and extensions may differ drastically in size, cost, and performance, but every program written for the native architecture can run on the subset or extended machine.

The Interdata 8/32 had difficulty satisfying this criterion since it has variable-length instructions, and there is no way after a trap or an interrupt to tell whether the instruction that was being executed was a 16-, 32-, or 48-bit instruction. This may be a problem when it is desired to correct the cause of the fault, and then reexecute (or resume) the instruction. Due to uncertainties in the definition of the Interdata 8/32 architecture, the CFA committee was not able to resolve whether or not the Interdata 8/32 satisfied this criterion.

e. Subsetability

At least the following components of an architecture must be able to be factored out of the full architecture:

- Virtual-to-physical address translation mechanism
- Floating-point instructions and registers (if separate from general purpose registers)
- Decimal instruction set (if present in full architecture)
- Protection mechanism.

Implementations of the architectures on small machines for dedicated applications must not be required to include features of the architecture intended for use on larger, multiprogrammed, multiapplication configurations. Existence of such subsets did not have to be demonstrated in an operational implementation of the architecture.

Because there was no operational method for testing subsetability, we could not challenge positive replies for any of the nine candidate architectures. However, the B-6700 and the AN/UYK-7 have not been designed for subsets in the sense of the criterion, so that their subsetability is more speculative.

In order to retain program compatibility for all implementations of the architecture, this criterion was extended to include the following requirement: *The trap mechanism of the architecture must be defined such that instructions in the full architecture, but not implemented in the subset machine, trap on the subset machine and that it be possible to write trap routines for the subset machine that allow it to interpretively execute those instructions not implemented directly in hardware (or firmware) and then resume execution.* (This is an elaboration of absolute criterion 4.)

f. Multiprocessor support

The architecture must support some form of "test-and-set" instruction to allow for the communication and synchronization of multiple processors. The intent of this criterion is to be sure that the basic architecture can support multiprocessor configurations.

g. Input/output controllability

A processor must be able to exercise absolute control over any I/O processor and/or I/O controller. The interpretation of the criterion proved rather difficult. Although all architectures necessarily permitted individual devices to be started and queried for status, there were varying degrees of control exercisable with respect to stopping the devices. It is reasonable to stop all I/O functions, or to stop selected devices. All architectures had some way of stopping a single device and stopping all devices, but how they did it varied widely in efficiency.

h. Extensibility

The architecture must have some method for adding instructions to the architecture that are consistent with existing formats. There must be at least one undefined code point in the existing opcode space of the instruction formats. All nine candidate architectures have unused instructions, so all passed this criterion.

i. Read-only code

It must be possible to execute programs from read-only storage. It was intended that this criterion permit an added degree of reliability by permitting programs to be stored in a nonvolatile read-only memory. However, a program can be rewritten to be read-only on any one of the nine architectures even if that architecture does not support special types of instructions to facilitate this. It might have been more meaningful to examine this question quantatively.

Table III-3 shows the score for each candidate architecture on each absolute criterion. Note that none of the nine architectures failed to meet the last five criteria: subsetability, multiprocessor support, I/O controllability, extensibility, and read-only code. This is in part the case because we limited our evaluation to reasonably successful architectures, but it is partly the result of not defining these criteria precisely enough prior to applying them to the candidate architectures. For example, by not clearly defining how to test for the practical subsetability of

Table III-3 — Candidate CFA Values for Quantitative Criteria

	Quantitative Criteria	Candidate CFAs								
		IBM S/370	Inter-Data 8/32	Rolm AN/UYK-28	Dec PDP-11	Univac AN/UYK-7	SEL 32	Burroughs B6700	Univac AN/UYK-20	Litton AN/GYK-12
1	V_1^*	27	27	20	20	24	22	24	20	20
2	V_2^*	27	27	20	19	24	22	20	17	20
3	P_1^*	27	27	22‖	25	23	26†	24	20	29
4	P_2^*	27	27	22‖	24	23	26†	20	17	29
5	U	0.371	0.355	0.039	0.043	0.15	0.450	0.019	0.125	0.219
6	C_{S1}	1344	1632	1008	1168	992	304	306	1328	1008
7	C_{S2}	576	576	112	144	448	288	204	336	752
8	CM_1	3168	1120	1882	736	1472	768	408	2256	1344
9	CM_2	1312	32	544	480	1472	704	408	720	1088
10	K	1	0	0	1	0	0	0	0	0
11	B_1	17,300	185	38,000‡	14,700	346	75	90	400	30
12	B_2	16,000	14	169	311	147	23	207	8	6
13	I	64	16	48	16	128	64	169	80	32
14	D	15	27	20	19	18	22	18	20	20
15	L	6192	560	114	112	2112	288	255	—	1376
16	J_1	1904	2368	1360	1040	1280	960	459	1408	1344
17	J_2	1136	1280	320	400	1280	960	459	640	1088
TAI	(%)	83	31	44	48	43	34	46	29	28
C_{SI}	($)	10.01	28.57	19.71	18.00	20.13	26.54	19.13	31.06	32.42

Note: Table adopted from Ref. 18.

*These values are of the form 2x where x = indicated data except for B6700, which is of the form 3(2x).

†With memory bank switching.

‡Include Novas.

‖Millions of dollars.

an architecture, we made it virtually impossible for an architecture to fail this criterion. Subsequent studies would be well advised to consider more precise definitions of these (and any additional) absolute criteria before evaluating alternative architectures against them.

2. Quantitative Criteria

In addition to the absolute criteria, the CFA committee specified 17 quantitative criteria that they felt would be helpful in the initial screening process. A number of these quantitative criteria measure attributes of a computer architecture better measured by benchmarks, or test programs. However, the CFA committee recognized that it did not have the resources to run benchmarks on all nine candidate architectures and therefore proceeded with the use of these quantitative criteria to help select three or four candidate architectures out of the original nine candidate architectures for more intensive study via test programs [18, 21].

The quantitative criteria are described below, and the score for each architecture on the quantitative criteria is given in Table III-3.

a. Virtual address space

V_1: The size of the virtual address space in bits.
V_2: Number of addressable units in the virtual address space.

Two aspects of these measures were open to interpretation. The CFA committee settled on the following interpretation for treating bank switching: the virtual address for a machine with bank switching is the address within a bank. The effect of bank switching is to increase the size of the physical rather than the virtual address.

The second interpretation centered on the notion of "addressable unit." There are several degrees of addressablity. An item may be fully addressable in the sense that it can be accessed by the address produced by an effective address computation. The committee also decided, however, that instructions such as the IBM S/370 Test Under Mask, and the OR Immediate allowed the testing and setting of individual bits, and provided a minimum addressable unit of 1 bit.

b. Physical address space

P_1: The size of the physical address space in bits.
P_2: The number of addressable units in the physical address space.

Where bank switching has been implemented, the physical address measures include all banks of memory available. For computers with virtual address translation, the physical address is the address resulting from the virtual-to-physical address translation. The physical address space is defined apart from any implementation, since the physical address space size is defined by the effective address calculation process or the virtual address translation process and need not be equal to the largest memory configuration yet delivered.

c. Fraction of instruction space unassigned

It is important to select an architecture that will allow reasonable growth over its expected lifetime. Let U be defined as the fraction of the instruction space in the architecture that is unassigned. Specifically,

$$U = \sum_{1 \leqslant i < \infty} u_i 2^{-i},$$

where u_i is the number of unassigned instructions of length i.

d. Size of central processor state

The amount of information that must be stored or loaded upon interrupt and/or context swapping is clearly an important factor in the response of real-time systems and in the overhead of multiprogramming systems. Let the processor state be defined as all the bits of information in a processor that must be saved in order to be able to restart an interrupted process at a later date. Processor states normally include the accumulators, index registers, program counter, condition codes, memory mapping registers, interrupt mask registers, etc.

C_{s1}: The number of bits in the processor state of the full architecture.

C_{s2}: The number of bits in the processor state of the minimum subset of the architecture (i.e., without floating-point, decimal, protection, or address translation registers).

C_{m1}: The number of bits that must be transferred between the processor and primary memory to first save the processor state of the full architecture upon interruption and then restore the processor state prior to resumption. This measure differs from C_{s1} above in that "register bank switching," where provided for in the candidate architectures, may eliminate the need to save some registers in primary memory, while the instruction fetches required to save the state are included in C_{m1} but not in C_{s1}.

C_{m2}: The measure analogous to C_{m1} for the minimum subset of the architecture.

These measures give an approximation of the complexity of the implementation of the architectures, as well as a measure of the responsiveness of the architectures to worst-case context changes for interrupt processing.

If an architecture provides for several sets of certain registers to provide fast switching or multiple contexts, and if a program uses only one such register set when it runs in one context, then only one set of these registers is used in calculating C_{s1}.

e. Usage base

B_1: Number of computers (delivered) as of the latest date for which data exist prior to June 1, 1976.

B_2: Total dollar value of the installed computer base as of the latest date for which data exist prior to June 1, 1976.

These two measures are meant to be approximate indicators of the existing software and programmer experience base. A single individual determined the value of these measures for all candidate architectures from standard sources.

f. I/O initiation

I: The minimum number of bits that must be transferred between main memory and any processor (central or I/O) in order to output one 8-bit byte to a standard peripheral device.

Although this measure was intended to give some insight into the responsiveness of an architecture, it is very difficult to construct an interpretation of the measure that serves this purpose well. The measure counts relatively few bits for some architectures, and this, in turn, makes the measure very sensitive to changes of a few bits. The I measure is also sensitive to several assumptions about exactly what actions are to be performed in doing the input/output operation, and where parameters for the operation are found. Unfortunately, this sensitivity made the I measure very arbitrary, and a rather inexact measure of input/output responsiveness.

g. Virtualizability

K: is unity if the architecture is virtualizable as defined in Ref. 6; otherwise, K is zero.

The intent of this criterion is to capture the concept of virtual machines that has been used to advantage in some commercial computer systems (e.g., IBM's VM/370). An architecture that supports virtual machines provides a mechanism for a privileged, stand-alone program to run as an unprivileged task and produce the results identical to those it produces as a privileged program. The importance of this idea is that an operating system can be run in user mode as a subsystem of another operating system.

The definition of virtual machine as provided by Popek and Goldberg [22] is a very strict definition that guarantees that any operating system that can run stand-alone on architecture X, can also run on architecture X in nonprivileged mode. If an architecture fails this definition it may still support virtual machines in a more limited sense.

h. Direct instruction addressability

D: *The maximum number of bits of primary memory that one instruction can directly address given a single base register, which may be used but not modified.*

Large displacement fields in instructions generally simplify programming because they reduce the need to set base registers and to maintain addressability. Because an architecture may have several different instruction formats, each with different displacement field formats, the committee required that the format selected for this measure be the one used for standard LOAD and STORE operations, or the equivalent thereof. This eliminated anomalies, like the MOVE CHARACTER LONG in the IBM S/370 architecture, from consideration.

i. Maximum interupt latency

Let L be the maximum number of bits that may need to be transferred between memory and any processor (central processor, I/O controller, etc.) between the time an interrupt is requested and the time that the computer starts processing that interrupt (given that interrupt are enabled). This may be interpreted as a measure of the longest noninterruptible instruction or sequence of instructions. Architectures with nonterminating, noninterruptible instructions have infinite L measures and are so indicated in Table III-3.

j. Subroutine linkage

J_1: The number of bits that must be transferred between the processor and memory to save the user state, transfer to the called routine, restore the user state, and return to the calling routine, for the full architecture. No parameters are passed.

J_2: The analogous measure to S_1 above for the minimum architecture (e.g., without floating-point registers).

This measure gives an indication of the size of overhead that might be encountered in doing subroutine calls in the case for the biggest and smallest machines in the family. The bits counted here are related to the count in CS_1, CS_2, CM_1, and CM_2. By presumption, the bits that are stored for J_1 are exactly those for CS_1, except that it is not necessary to save the protection registers, memory map registers, interrupt mask, and other registers that determine the global context for a program. Architectures with small processor states or that have LOAD/STORE MULTIPLE instructions show up well on these measures.

C. Summary

As introduced, the architecture of a computer is complicated by the many aspects from which it may be viewed. This section has presented two such aspects: (a) hardware architecture and (b) software architecture. The hardware architecture is characterized by ten elements that may be used to distinguish one configuration from another in performance. In a similar manner the software architecture is described by nine qualitative features and 17 quantitative features. For the most part, these architectures are manufacturer oriented. The quantitative features, when used to evaluate a particular architecture, provide a basis for the relative (and absolute) performance of a computer architecture. In the ensuing sections, these criteria will provide the basis to evaluate, specify, and select computers for requirements as discussed in the previous section.

For ESM applications where throughput is the predominant factor, memory access time T_m tends to dominate other considerations. In software, costs, which bear a high correlation to the total architectural inventory dollar value B_2, combine to provide the basis for a cost-effectiveness evaluation of the particular computer architecture.

Finally, another architectural facet not addressed is technology, which appears to transcend the above. Hardware architecture can (and should) survive technological advances that would improve throughput. Software, too, can be designed so as to take advantage of improved technology without requiring change.

IV. EVALUATION

Preliminary to and necessary for the specification of a computer processor, is the ability to evaluate such a computer. This section develops the quantitative criteria for evaluation. Having done so, it is possible to understand how specification is related quantitatively to the data requirements of the system that the computer/processor must support. Again, as in the architecture, there is a hardware and software aspect to this evaluation. The hardware evaluation develops those factors that most represent throughput of an (EW/ESM) processor. These factors are then applied to the actual characteristics of some of the more common military and commercial computers to evaluate their performance in an EW/ESM system. These indices of performance, which are in fact absolute for the computer, are then compared with their software performance as determined in the CFA/MCF studies. In the latter comparisons, the software factors that contribute most to the EW/ESM processing are isolated by the hardware performance index, and the software architectural criteria are quantitatively weighted for their significance to EW/ESM.

Table IV-1 summarizes the various efforts at processor architecture evaluation, both hardware and software. In some cases the figures are induced by cross-correlation as indicated. The hardware column represents the ESM weighted average instruction-execution time. As such it represents the absolute throughput of that architecture to process ESM data. The smaller number implies a faster throughput. The next four columns are software architecture related and evaluate the processor architectures for their efficiency in general military processing. Again in these columns the smaller value indicates a more efficient processing architecture.

A. Hardware

One goal in an ESM System design is to select a complement of components and techniques that will maximize the rate at which the emitters in the environment can be processed and identified without error. Various techniques were presented in the previous section to increase the efficiency and the speed of an ESM processor. These techniques were generally aimed at the designer who intends to specify or build a special purpose processor to do the ESM processing job.

This section will address the factors that must be considered when a general purpose computer is used to perform the processing task.

1. Main Computer

Since one of the most important goals of an ESM System is to increase the emitter collection and identification throughput rate, it follows that one of the most important criteria for evaluating an ESM computer is the relative computation speed.

There are two diverse approaches that may be taken to evaluate the processing speed of a microcomputer in the ESM environment. The first approach is to actually encode an entire ESM Analysis Program in computer language, and then make a detailed analysis of the system throughput. This approach gives a very accurate estimate of the throughput capability of a computer, but it is very time-consuming and, therefore, very expensive.

Table IV-1 — Evaluation Results
(Lower is Better)

Manufacturer	Computer Processor	Related Architecture	(EW)‖ Hardware	CFA Phase ϕ	CFA Phase I(M)	CFA Phase II(M)	CFA Support Software	Real-Time Throughput (Pulse words/s)	TAI
IBM	S/370		2.182*	0.735	1.27			11079	73%
Interdata	A/32		2.235*	0.595	0.85			10817	34%
Nova/Rolm	AN/UYK-28	AN/UYK-19	2.108	1.087		1.19	0.861	11468	42%
Dec	PDP-11	AN/UYK-21	0.944b 2.293c	0.699	0.93	0.88	0.572	25609b 10453	48%
Univac	AN/UYK-7		1.174†	2.174		1.38	0.615	20592	59%
SEL	SEL-32		2.019*	1.163				11974	(34%)
Burroughs	B6700		2.019*	1.163				11974	(46%)
Univac	AN/UYK-20	AN/AYK-14	1.746	2.273		0.73	1.206	13846	(29%)
Litton	AN/GYK-12		1.525†	1.064		0.96	1.746	15852	21%
IBM	AN/AYA-6	4π	4.980					4854	
ATI	ATAC-S	ATAC-F	2.668					9061	
ATI	ATAC-F	ATAC-S	1.163					20787	
Univac	AN/UYK-15	AN/UYK-10	1.586					15243	30%

*Architecturally based

†Benchmark based

‡Microinstruction based

‖mean $\simeq$ 2.048; $\sigma = \pm$ 0.978.

The second approach is to estimate the system throughput based on a weighted average of the computer's instruction execution times. This may be accomplished in the following manner:

A representative ESM Analysis Program is selected and encoded, using an instruction set that has been optimized for ESM processing. The total number of times that each instruction is used is tabulated, taking into account all program loops and all repetitions of each subroutine. This gives the percentage use or the weight factor for each instruction. When these weight factors are multiplied by the time required to perform each instruction on a given computer, a quantitative rating of the relative throughput capability of that computer is achieved. The steps required for this technique are outlined below.

1. Selected a representative ESM Processing System.

2. Chose the primary (most used) instructions (About 10 percent of the instructions should perform over 50 percent of the processing.)

3. Determine the weight factor. (This is the precent of time that each primary instruction is used.)

4. Determine the time required to perform each primary instruction on the given processor.

   On microprogrammable computers: microprogram the instruction set into the computer.

   On regular computers: If a given instruction is not available, write a "macro" subroutine.

5. Multiply the time per primary instruction by the weight factor and sum for all primary instructions to get the relative rating.

The disadvantage of this technique is that the various computers use different instruction sets. Although most instructions in the various sets are equivalent to one another, some way must be found to adjust for the variances.

There are no such difficulties when microprogrammable computers are compared, since each computer can be microprogrammed to have the complete instruction set required for the evaluation. Once the instructions have been microcoded, the total operating time for each instruction can be readily found by multiplying the number of microinstructions required by the microinstruction cycle time.

The advantage of using the weighted instruction set approach is that, when the weights are actually calculated for the various instructions, it is found that 10 percent of the instruction set is used to perform about 60 percent of the processing, while the least-used 40 percent of the instruction set does only about one percent of the processing. As a result, the infrequently used instructions can be ignored without significantly affecting the rating of the processor.

A very good estimate of the ESM processor capability can, in fact, be made by considering only the most-often used 19 percent of the instruction set. This produces a reasonably quick

method for sorting through a large number of available computers and ending up with a few top-rated machines that could then be subjected to the detailed evaluation discussed previously.

a. Determining ESM weight factors

For determining the instructions that are used most often in ESM processing and for determining the weight factors, a "representative" ESM processing program is needed. The program selected here is the EXCAP ESM Program, which is used to do the processing aboard the EA-6B.

The best method for determining the weight factors is to estimate the relative number of times each functional subprogram is run during the normal operation of the system. Then, the number of times each instruction is used during normal operation of each subprogram is tabulated, taking into account each repetition of each instruction within all the program loops found in that subprogram. Unfortunately, the number of times each subprogram is run, and the number of times each of the loops is iterated within the subprogram, are strongly dependent on the signals present in the environment at any given time. The detailed determination of the instruction usage in a real operating system requires a great deal of statistical analysis of the interaction of the system with the expected signal environment. To simplify the analysis, all program loops and subprogram repetitions are ignored, and each instruction is counted only once each time it appears in the EXCAP Program listing.

The EXCAP Program contains a total of 103 separate, distinct assembly language instructions (not counting the different addressing modes). These are listed in Table IV-2 in the order of their frequency of usage. It is apparent from the table that the first 21 instructions, representing the most-used fifth of the instruction set, do about 84 percent of the total processing, while the least-used 51 instructions, almost half of the instruction set, perform only 2.4 percent of the processing. Clearly, ignoring these last 51 instructions would have a negligible effect on the evaluation of the relative processing speed of a given computer.

The main factors that determine computer throughput (such as the processor's microcycle time, the memory access time, whether the computer uses pipelined architecture, etc.) have a similar influence on the execution times of all the computer's instructions. Therefore, a good estimate of the relative throughput of various computers can be achieved by considering only those few instructions that do the bulk of the processing.

The 21 most-used instructions are shown in Table IV-3, along with their percentage contribution to the total EXCAP program and the cumulative percentage contribution to the program. The same data are plotted in Fig. IV-1 to provide a more readily understandable graphic presentaton of the data.

The weights are simply the fraction of the number of times a given instruction is used out of the total instruction set used for calculating weights. Therefore,

$$w_i = \frac{f_i}{\sum_{j=1}^{N} f_j}, \qquad \text{(IV-1)}$$

Table IV-2 — Assembly Language Instruction Usage in the EXCAP Program

Instruction	Frequency	Instruction	Frequency	Instruction	Frequency	Instruction	Frequency
LAH	2073	BM	73	BSM	11	CUT1	1
MIC	1598	IOSE	72	RSPB	11	GPLM	1
SAH	1569	MH	66	SNZ	11	NLM	1
B	1043	SLF	57	ISPB	10	SEOM	1
BSI	834	BOC	48	SNM	8	SEOP	1
LXR	665	D	47	SS	7	SNP	1
SA	500	MFTO	47	SZ	7	SP	1
LA	477	XOR	46	LXI	6		
MXR	454	SRAD	43	SM	6	Total	
SBZ	447	BNM	40	SPLM	6	Statements	15,365
DIOC	416	LSW	40	COM	5		
AUA	351	BP	39	XORS	5		
SR	349	AN	38	BCN	4		
SH	343	LXA	37	PMC	4		
MSH	322	SRA	32	ALE	3		
SXR	277	ALA	31	BSZ	3		
SL	267	BSNZ	31	CUT2	3		
AH	263	CVD	31	IOCR	3		
BZ	228	AUE	28	MFT1	3		
SHZ	224	ANS	27	BC	2		
SHN	197	SQ	26	BSNM	2		
BNZ	181	BNP	25	BSOM	2		
M	171	AS	23	CUT3	2		
ORS	138	OR	22	SE	2		
A	119	USIN	21	SLCD	2		
AUO	110	CVB	20	XAS	2		
CH	95	ALO	18	BCON	1		
SLD	90	C	16	BOP	1		
S	87	IOCW	14	BSNP	1		
IORS	82	SLC	13	BSOP	1		
SRL	82	SRRD	13	BSP	1		
LQ	74	BO	12	CP	1		

Table IV-3 — EXCAP Program Instruction Usage
by Individual Instruction

Instruction	Frequency	Percent	Cummulative Percent	Comment
01. LAH	2073	13.49	13.5	Load accumulator half-word
02. MIC	1598	10.40	23.9	Modify instruction counter
03. SAH	1569	10.21	34.1	Store accumulator half-word
04. B	1043	6.79	40.9	Branch
05. BSI	834	5.43	46.3	Branch to subroutine
06. LXR	665	4.33	50.6	Load index register
07. SA	500	3.25	53.9	Store accumulator
08. LA	477	3.10	57.0	Load accumulator
09. MXR	454	2.95	60.0	Modify index register
10. SBZ	447	2.91	62.9	Skip if bit is zero
11. DIOC	416	2.71	65.6	I/O control
12. AUA	351	2.28	67.9	"AND" upper half-word
13. SR	349	2.27	70.1	Shift right
14. SH	343	2.23	72.4	Subtract half-word
15. MSH	322	2.10	74.5	Modify storage half-word
16. SXR	277	1.80	76.3	Store index register
17. SL	267	1.74	78.0	Shift left
18. AH	263	1.71	79.7	Add half-word
19. BZ	228	1.48	81.2	Branch on zero
20. SHZ	224	1.45	82.7	Skip if half-word is zero
21. SHN	197	1.28	83.9	Skip if half-word is negative

Note: Total: 103 instructions, 15,365 statements.

where

ω_i is the weight of the ith instruction.

f_i is the frequency of the ith instruction.

N is the number of instructions used in the evaluation.

The average time to execute an instruction is the weighted average of the individual instruction execution times:

$$T_{av} = \sum_{i=1}^{N} w_i t_i \qquad \text{(IV-2)}$$

where t_i is the time required to perform the ith instruction (in microseconds).

The relative throughput rating of the microcomputer is an estimate of the number of instructions the computer can execute per second:

$$R = \frac{1}{T_{av}} = \frac{1}{\sum_{i=1}^{N} w_i t_i}. \qquad \text{(IV-3)}$$

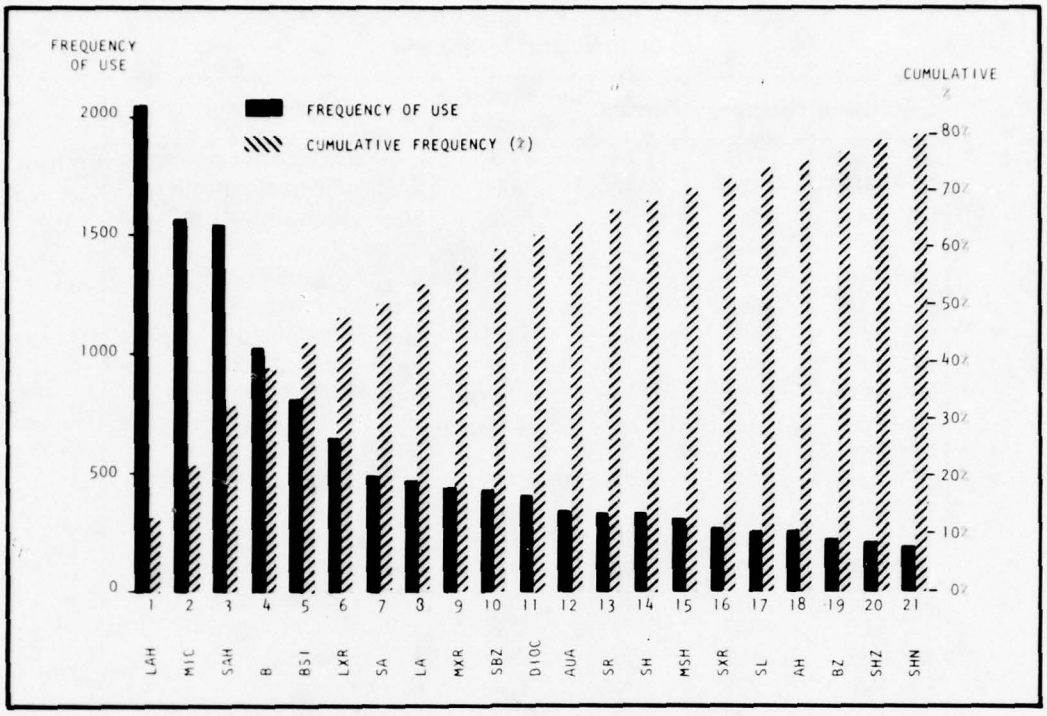


Fig. IV-1—ESM program instruction usage by individual instruction

b. ESM ratings of computers

The relative ratings of computers were calculated with the aid of the above formulas, using the 21 most-used instructons of the EXCAP instruction set. The results are shown in Table IV-4.

The computers evaluated were the AN/AYA-6 (4π) computer used in the EXCAP System, two versions of Applied Technology's ATAC computer, the AN/UYK-15, AN/UYK-20, AN/UYK-23, and AN/AYK-14 computers [13,15-17,23].

The ATAC computers both use a 2900-type bit-slice microprogrammable microprocessor with a basic cycle time of 250 ns. The computers are identical in every way, except for the memory modules. The ATAC-S uses an MOS memory with a 1-μs cycle time (650-ns access time), and the ATAC-F uses a bipolar, random-access memory with a 200-ns cycle time. The AN/UYK-15 and AN/UYK-20 both use a split-memory architecture with a memory read/write cycle time of 750 ns. The AN/UYK-28 memory is magnetic core with an access time of 1 μs, but it uses special purpose, high-speed hardware to do some of the processing. Since the EXCAP instruction set used was written for the AN/AYA-6 computer, the other computers are penalized slightly by sometimes having to perform several instructions for one EXCAP instruction.

Table IV-4 — Relative Ratings of Computers Based on the 21 Most-Used Instructions that do 84% of the Processing

Instruction	Weight	AN/AYA-6	ATAC-S*	ATAC-F*	AN/UYK-15	AN/UYK-20†	AN/UYK-28	AN/AYK-14†
LAH	0.1607	0.804	0.442	0.2210	0.2411	0.2411	0.402	0.2250
MIC	0.1239	0.413	0.155	0.0620	0.1487	0.1363	0.124	0.1735
SAH	0.1217	0.659	0.304	0.1521	0.1826	0.2069	0.304	0.2677
B	0.0809	0.303	0.283	0.1011	0.1456	0.1375	0.129	0.1052
BSI	0.0647	0.386	0.348	0.1294	0.1068	0.1876	0.233	0.1359
LXR	0.0416	0.172	0.103	0.0258	0.0774	0.0774	0.114	0.0929
SA	0.0388	0.226	0.136	0.0679	0.0873	0.0931	0.163	0.0737
LA	0.0370	0.185	0.139	0.0648	0.0833	0.0833	0.155	0.0888
MXR	0.0352	0.142	0.053	0.0132	0.0528	0.0528	0.035	0.0510
SBZ	0.0347	0.195	0.134	0.0434	0.0850	0.1006	0.069	0.0798
DIOC	0.0323	0.337	0.081	0.0565	0.0969	0.1454	0.116	0.1131
AUA	0.0272	0.091	0.054	0.0136	0.0408	0.0408	0.027	0.0488
SR	0.0271	0.237	0.088	0.0678	0.0894	0.0461	0.087	0.1653
SH	0.0266	0.133	0.053	0.0150	0.0399	0.0399	0.027	0.0386
MSH	0.0250	0.177	0.156	0.0719	0.0938	0.0988	0.075	0.1125
SXR	0.0215	0.134	0.075	0.0323	0.0484	0.0516	0.048	0.0409
SL	0.0207	0.285	0.067	0.0518	0.0683	0.0352	0.066	0.0642
AH	0.0203	0.102	0.041	0.0114	0.0305	0.0305	0.020	0.0294
BZ	0.0176	0.084	0.051	0.0176	0.0317	0.0299	0.035	0.0308
SHZ	0.0173	0.101	0.032	0.0130	0.0208	0.0190	0.035	0.0242
SHN	0.0152	0.089	0.029	0.0114	0.0182	0.0167	0.030	0.0213
Factor								
T_{av}		5.255	2.824	1.2430	1.7893	1.8705	2.2940	1.9826
R		0.190	0.354	0.805	0.559	0.535	0.436	.504
R/R_1		1.00	1.86	4.23	2.94	2.81	2.29	2.65

*Related architecture, different memory speeds.
†Related architecture.

Since the actual addressing modes used by the program were not considered, the timing used for this evaluation was the one for indexed addressing (the most prevalent mode used). Since this is the fastest addressing mode, the execution times listed are slightly faster than they would have been if all the different addressing modes had been considered. Nevertheless, the results of the comparison serve to illustrate the differences between the computers.

When using a partial instructon set to rate computers, there is always the question of where to draw the line. Certainly 21 instructions are much easier to compare than 103 instructions, but the question remains whether fewer instructions would give a satisfactory comparison with much less effort. Looking at Table IV-3, note that just six instructions perform half the total programming. Therefore, an evaluaton of the operating speed of just these six instructions should give a good estimate of a computer's relative throughput.

The comparison of the six computers, based on the six instructions which do 50 percent of the processing, is presented in Table IV-5. If only six instructions are used, the average instruction execution time is somewhat underestimated. The reason for this is that some of the most-used instructions run relatively fast—this is especially true of the MIC instruction—while the slower, more complex instructions are used less often. Nevertheless, the difference

Table IV-5 — Relative Ratings of Computers Based on the Six Most-Used Instructions That Do 50 Percent of the Processing

Instruction	Weight	AN/AYA-6	ATAC-S*	ATAC-F*	AN/UYK-15	AN/UYK-20†	AN/UYK-28	AN/AYK-14†	PDP-11/45* Bipolar Memory	PDP-11/45* Core Memory
LAH	0.267	1.34	0.734	0.367	0.401	0.401	0.668	0.374	0.280	0.742
MIC	0.205	0.68	0.256	0.103	0.246	0.226	0.205	0.287	0.123	0.232
SAH	0.202	1.09	0.505	0.253	0.303	0.343	0.505	0.444	0.212	0.562
B	0.134	0.50	0.469	0.168	0.241	0.228	0.214	0.174	0.121	0.192
BSI	0.107	0.64	0.575	0.214	0.177	0.310	0.385	0.225	0.161	0.281
LXR	0.085	0.28	0.170	0.043	0.128	0.128	0.187	0.153	0.089	0.236
Factor										
T_{av}		4.53	2.71	1.15	1.496	1.636	2.164	1.657	0.986	2.245
R		0.221	0.369	0.871	0.668	0.611	0.462	0.604	1.014	0.445
R/R_1		1.0	1.67	3.94	3.03	2.77	2.09	2.733	4.588	2.016

*Related architecture, different memory speeds.

†Related architecture.

between the rating based on six instructions and the rating based on 21 instructions is quite small; using the six most-used instructions, which perform 50 percent of the processing, gives a fairly good indication of the relative throughput of various computers.

c. Rating by instruction category

In most computers there is a large number of instructions that are executed identically and which have identical execution times. In a computer with a number of general purpose registers, for example, the time required to load any given register is usually the same as the time required to load any other register. Add and subtract instructions usually follow the same signal flow paths and have the same execution times, the only difference being the setting of the arithmetic logic unit. Similarly, logical expressions such as AND, OR, NAND, etc., usually take the same time to execute.

A considerable saving of effort can be made in the weight-factor method of evaluating computers if all the instructions that are executed identically are collected into instruction categories, and the weights are calculated by category rather than by individual instructions.

Taking the EXCAP Program again as an example, the EXCAP instruction set of 103 instructions was arranged into 40 separate categories. As was the case with the individual instructions, it was found that the most-used 10 instruction categories accounted for 79 percent of the total processing done in the program. These 10 instruction categories are listed in Table IV-6 in the order of the frequency of usage. The same data are illustrated in Fig. IV-2.

The six computers that were rated previously are rated again by instruction categories. The ratings, shown in Table IV-7, are based on the 10 instruction categories which account for 79 percent of the total processing. The computers were also compared, based on the top four instruction categories, which account for 54 percent of the processing. The results are presented in Table IV-8.

Table IV-6 — ESM Program Usage by Instruction Category, Ranked by Frequency of Use

Instruction Category	Frequency	Cumulative Percent
1. Load Internal Register	3289	21.4
2. Store Internal Register	2372	36.8
3. Quick Branch	1598	47.2
4. Branch	1043	54.0
5. Branch to Subroutine	834	59.5
6. Register Add/Subtract	812	64.7
7. Branch on Condition	622	68.8
8. I/O Control	587	72.6
9. Register Logical Operation	541	76.1
10. Modify Internal Register	454	79.1
Total — 10 Categories	15,365 Instructions	

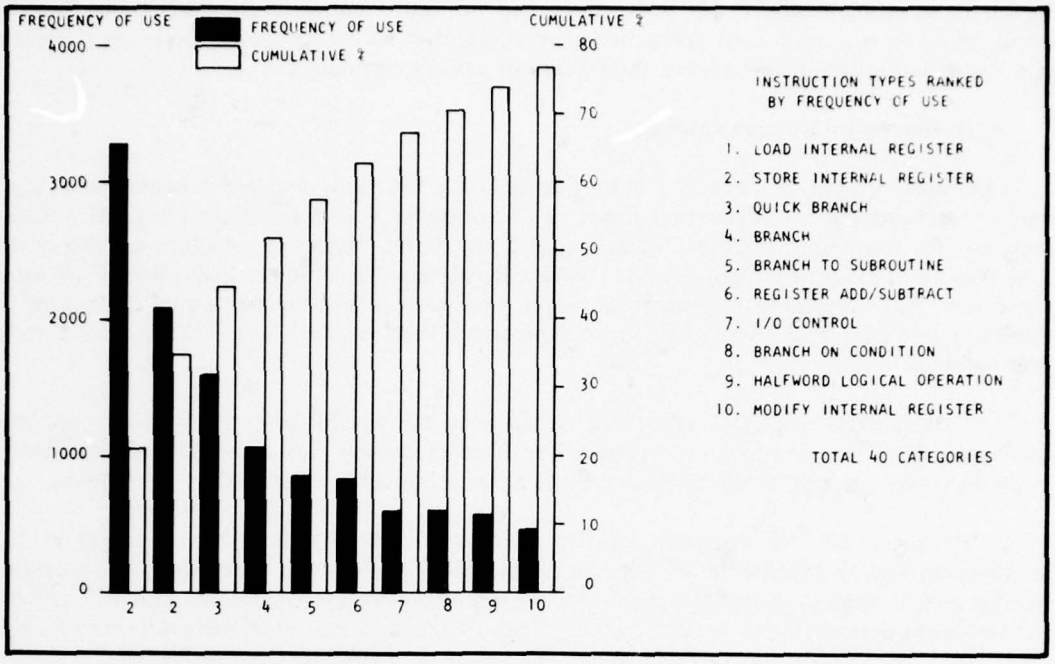


Fig. IV-2—ESM instruction usage by instruction category

Table IV-7 — Relative Ratings of Cumputers Based on the Ten Most-Used Instruction Categories That Do 79 Percent of the Total Processing

Category	Weight	AN/AYA-6	ATAC-S	ATAC-F	AN/UYK-15	Univac AN/UYK-20	Rolm AN/UYK-28
LIR	0.271	1.355	0.745	0.373	0.407	0.407	0.678
SIR	0.195	1.076	0.536	0.268	0.293	0.332	0.488
QBR	0.132	0.440	0.65	0.066	0.158	0.145	0.132
BR	0.086	0.323	0.301	0.108	0.155	0.146	0.138
BSI	0.069	0.412	0.371	0.138	0.114	0.200	0.248
Reg A/S	0.067	0.335	0.134	0.038	0.101	0.101	0.068
BOC	0.051	0.244	0.147	0.051	0.092	0.087	0.102
I/O	0.048	0.500	0.120	0.084	0.144	0.216	0.173
Logical	0.045	0.150	0.090	0.023	0.068	0.068	0.045
MIR	0.036	0.145	0.054	0.014	0.054	0.054	0.036
Total		4.980	2.668	1.163	1.586	1.746	2.108
Rating		0.201	0.376	0.860	0.631	0.573	0.474
R/R_1		1.00	1.87	4.28	3.14	2.35	2.36

Table IV-8 — Relative Ratings of Computers Based on the Four Most-Used Instruction Categories That Do 54 Percent of the Processing Instructions

Category	Weight	AN/AYA-6	ATAC-S*	ATAC-F*	AN/UYK-15	AN/UYK-20†	AN/UYK-28	AN/AYK-14†	PDP-11/45* Bipolar Memory	PDP-11/45* Core Memory
LIR	0.396	1.980	1.089	0.545	0.594	0.594	0.990	0.554	0.416	1.101
SIR	0.286	1.579	0.737	0.393	0.429	0.486	0.715	0.629	0.300	0.795
QBR	0.192	0.640	0.240	0.096	0.230	0.211	0.192	0.269	0.115	0.217
BR	0.126	0.473	0.441	0.158	0.227	0.214	0.202	0.164	0.113	0.180
Total Rating		4.672	2.557	1.192	1.480	1.505	2.099	1.616	0.944	2.293
R/R_1		1.0	1.83	3.92	3.16	3.10	2.23	2.892	4.950	2.036

*Related architecture, different memory speeds.
†Related architecture.

Comparing Tables IV-4 through IV-8 allows one to reach some conclusions about the computer evaluation technique and the relative rating of computers:

- Tables IV-4 and IV-8 are expected to give the best indication of the relative rating of the several computers. Certainly the rating value should be a close (but optimistic) estimate of the processor's throughput.

- The throughput ratings are optimistic because the especially slow computer instructions, which are not often used, are not included in the evaluation.

- The fewer the number of instructions included in the evaluation, the more optimistic the ratings become. The relative ratings, however, do not change very rapidly when the size of the data base is decreased.

Concerning the third item above, it should be noted that a remarkably good relative evaluation of various computers can be made by comparing the timing of the Load Accumulator (Load Internal Register) instruction alone. The relative ratings of the six computers based on the LA instruction are:

AN/AYA-6	—	1.0	AN/UYK-15	—	3.33
ATAC-S	—	1.82	AN/UYK-20	—	3.33
ATAC-F	—	3.64	AN/UYK-28	—	2.0 .

2. Preprocessor

The incoming data stream from the receiver is stored in a buffer, and signal characteristics such as pulse width, amplitude, TOA, frequency, and intrapulse amplitude and phase are determined.

The data are then fed to a signal sorter circuit, where the pulses from previously identified emitters and from unwanted emitters are removed from the data stream. The remaining signals, coming from interesting but unidentified emitters, are sent to a PRI processor and pulse deinterleaver secton where the PRI of the unidentified emitters is determined. The PRI data are multiplexed with the other emitter data in the signal formatter to form a set of emitter parameter words, which are then stored to await transfer to the main computer memory under DMA control. The parameter comparison function of the signal sorter is performed in real time at the maximum signal input rate from the environment. The input data rate could exceed a half-million data words per second. The rapid processing required can be performed only by dedicated hardware. Software processing for the signal sorting function with present-day processing equipment is impractical.

a. Signal sorter

The parameter comparison function of the signal sorter is performed in real time at the maximum signal input rate from the environment. The input data rate could exceed a half-million data words per second. The rapid processing required can be performed only by dedicated hardware. Software processing for the signal sorting function with present-day processing equipment is impractical.

b. Flywheel tracker

The simplest adaptive tracking technique, requiring a minimum of software and computing time, is to maintain a running average of the previously measured values of the parameter

estimates and a running average of the absolute value of the measured parameter deviations to set the limits. The equations for the predictor–corrector functions are given as:

$$\hat{S}_{p(k+1)} = \hat{S}_{pk} + \frac{\hat{S}_{mk} - \hat{S}_{pk}}{N} \tag{IV-4}$$

$$\hat{D}_{(k+1)} = \hat{D}_k + \frac{|\hat{S}_{mk} - \hat{S}_{pk}| - \hat{D}_k}{N} \tag{IV-5}$$

$$\hat{\sigma}_{k+1} = \left(M_1 + \frac{1}{M_2}\right) \hat{D}_{k+1} \tag{IV-6}$$

where

$S_{mk} - S_{pk}$ is the error signal from the comparator

D_k is the main absolute deviation

N, M_1 and M_2 are powers of 2.

This simple adaptive algorithm can serve as a benchmark to estimate the maximum signal-handling capabilities of various types of processors. Given $N = 16$, $M_1 = 4$, and $M_2 = -4$ (corresponding to the 3σ limit interval of a normal distribution), the algorithm can be performed in 32 assembly language instructions per parameter, including the calculation of $\hat{S}_m - \hat{S}_p$ and the upper and lower limits. Two additional instructions are required to update the TOA, so the entire update calculation (of three parameters) for one emitter requires 98 instructions, of which 32 instructions reference the memory.

The maximum pulse-to-pulse tracking capability for various processors is shown in Table IV-9, based on an assumed PRF of 1000 pps for each emitter. The table shows the approximate number of emitters that can be tracked when the tracking is based on three emitter parameters processed serially by a single processor. The number of emitters that can be tracked is tripled if each parameter is processed in parallel by three parallel processors. If the processors are microprogrammed to do the entire processing sequence with only one assembly instruction, all individual instruction fetch cycles are eliminated and the processing speed is increased by an additional factor of 3 or 4, depending on the type of memory used.

Table IV-9 — Signal Tracking Capability of Various Preprocessors
(Number of Emitters at 1000 Pulses per Emitter)

SOFTWARE					
Processor	Memory	General-Purpose		Microprogrammed	
		Series	Parallel	Series	Parallel
NMOS	NMOS	2	7	6	16
Bipolar	Core	6	18	17	46
AN/UYK-20		10			
Bipolar	Fast NMOS	11	31	24	66
Bipolar	Bipolar	17	50	30	85
HARDWARE					
Processor	Memory	Serial Memory		Parallel Memory	
		Series	Parallel	Series	Parallel
TTL	Core	36	94	133	400
TTL	Low-power TTL	105	333	350	1052
TTL	TTL	196	512	455	1395

Table IV-9 shows that a fast general-purpose computer, such as the AN/UYK-20, would spend its entire computing time in tracking only 10 emitters on a pulse-to-pulse basis, using the simplest possible tracking algorithms and ignoring the processing required to perform the flywheel function. It is quite obvious that the flywheel tracking function for any realistic environment cannot be implemented with a general-purpose software processor (computer), if the tracking has to be done on a pulse-to-pulse basis.

The designer has three options for implementing the processing required for a flywheel tracker:

- If a large number of emitters must be tracked, the tracking algorithm should be implemented in a hardwired logic processor.

- If the tracking is limited to a moderate number of emitters (under 30), and if simple processing algorithms can be used, the tracking and flywheel functions can be performed by parallel-organized dedicated, microprogrammed, bipolar processors that are specially designed to perform the tracking function on a pulse-to-pulse basis.

- If the tracking algorithms used are to be complex, the processing algorithms cannot be implemented on a pulse-to-pulse basis. In this case, it is best to perform the TOA update and the flywheel tracking with simple hardwired logic, and to use a parallel-organized microcomputer to periodically update the tracked emitter parameter file. Emitter parameters do not normally change very rapidly, so a few updates per second should be sufficient to maintain tracking, within wider tolerance limits, for most emitters.

A more complete evaluation of preprocessors is presented in Appendix A.

B. Software

Having applied the quantitative criteria discussed in the previous section, the CFA committee evaluated the performance of the candidate architectures on these criteria to screen out all but three or four of the architectures for further consideration in the test program and software evaluation phases of the study [18].

1. Phase ϕ

Clearly, the candidate architectures should be ordered relative to each of the 17 quantitative criteria, and these independent orderings then studied to detect weaknesses and strengths of the competing architectures. However, some summary measurements were ultimately needed to assist the committee in its selection of the final architectures to undergo more intensive study. Various thresholding and weighing schemes were proposed, but the particular scheme described here was the scheme chosen by the CFA committee.

a. Relative weighting of criteria

Each voting organization of the CFA committee was given 100 points to distribute among the various measures to indicate their relative importance to the organization. The weight for criterion x, $W[x]$, was defined as the total number of points given criterion x by all the voting

Table IV-10 — Quantitative Criteria Composite Weights*

Criterion	Army Weights	Navy Weights	Military Weights	EW Weights
V_1	0.0412	0.0444	0.0433	0.039
V_1	0.0438	0.0575	0.0529	0.000
P1	0.0425	0.0006	0.0612	0.045
P2	0.0387	0.0637	0.0554	0.022
U	0.0513	0.0644	0.600	0.051
CS1	0.0587	0.0375	0.0466	0.026
CS2	0.0675	0.0219	0.0371	0.090
CM1	0.700	0.0544	0.0596	0.014
CM2	0.0713	0.0319	0.0450	0.158
K	0.0500	0.0587	0.0558	0.000
B1	0.0450	0.0244	0.0313	0.128
B2	0.0200	0.0281	0.0254	0.063
I	0.0875	0.1419	0.1238	0.096
D	0.0912	0.1081	0.1025	0.048
L	0.0812	0.0969	0.0917	0.066
J1	0.0637	0.0626	0.0629	0.036
J2	0.0762	0.0331	0.0475	0.089

*After Ref. 18.

CFA organizations, divided by the total number of points handed out. The weights for the quantitative criteria based on responses from 24 voting CFA committee members are given in Table IV-10.

b. Normalization

When attempting to combine these quantitative measures into a composite measure there were two problems:

- The measures are defined such that good computer architectures maximize some measures and minimize others. Specifically, the measures that a computer architecture should maximize are V_1, V_2, P_1, P_2 U, K, B_1, B_2, and D; whereas the measures that should be minimized are CS_1, CS_2, CM_1, CM_2, I, L, J_1, and J_2.

The composite measure was a maximal measure and all minimal measures were transformed to maximal measures by taking the reciprocal, $X' = 1/X$.

- Measures that inherently involve large magnitudes were not necessarily more important than smaller measures. For example, V_1 was on the order of 10^4 to 10^9, while K was either 0 or 1.

To resolve this problem of differing scale, the values for the quantitative criteria were normalized by dividing each value by the average value of the criterion over the set of nine architectures. For eample, the nine measures for criteria I are 64, 16, 48, 16, 128, 64, 169, 80, 32; the average value is 68.6, and the normalized measures are 0.93, 0.23, 0.70, 0.23, 1.87, 0.93, 2.47, 1.17, 0.47.

Normalized measures have the attractive properties that they all lie in the range $(0,M)$; all have a mean across the set of M architectures of unity; and the standard deviation of the set of normalized measures is in the interval $(0,M^{0.5})$. We could have taken the normalization process a step further and adjusted the spread of each measure so that the measure gave a standard deviation of unity (or some other constant) across the set of architectures being evaluated. We did not do this for all measures. Some measures were better "discrimination functions" than others and we did not want in general to lose this information by further normalization. However, the committee agreed that it is important to normalize the standard deviation of some of the measures; specifically, V_1, V_2, P_1, P_2, and D were normalized to have a mean and standard deviation of unity. These measures may differ by several orders of magnitude between candidate architectures, but the CFA committee did not feel that the utilities, as expressed by the measures, differ by orders of magnitude.

c. Relative architecture scoring

In order to combine the individual measures, the committee used a simple, linear sum of each normalized measure x scaled by its corresponding weighting coefficient $W[x]$. The weighing coefficients have been defined so that they sum to unity and hence the composite measure A is in fact a normalized measure with a mean of 1. Using the weights given in Table IV-10 and the values of the quantitative criteria given in Section III, we get the composite measures for the candidate architectures shown below in Table IV-11.

Table IV-11 — Software Ranking Based on the Quantitative Criteria*

Architecture	Score	
	Military	EW
Interdata 8/32	1.68	2.235
PDP-11	1.43	2.293c
IBM S/370	1.36	2.182
AN/GYK-12	0.94	1.525
ROLM	0.92	2.108
B6700	0.91	2.019
SEL-32	0.86	2.019
AN/UYK-7	0.46	1.174
AN/UYK-20	0.44	1.746

*After Ref. 18.

There was some valid concern by members of the CFA committee about the role of the weighting of the measures, the normalization of the measures, and the measures themselves in the selection of finalists. However, upon detailed examination of the results we found that, given the weights applied by the committee as an indication of the importance of idealized concepts, the finalists selected are very insensitive to the exact details of the selection procedure. Almost any reasonable methodology for measuring the key concepts quantitatively would select the same finalists.

The scores for each of the candidate architectures are given for the absolute and quantitative criteria, respectively. Only the IBM S/370 and PDP-11 architectures passed all the absolute

criteria. The Interdata 8/32 architecture is not well defined with respect to trap handling, and there remains some question as to whether it meets the requirements of the interrupt and trap-handling criteria. The remaining six candidate architectures failed one or more of the absolute criteria specified by the CFA committee. A weighting scheme was developed by the CFA committee for the quantitative criteria, and the composite scores of the nine candidate architectures are given in Table IV-11. The quantitative criteria showed that the Interdata 8/32, PDP-11, and IBM S/370 lead the other architectures by comfortable margins. These results were used by the CFA committee to reduce the field of candidate architectures to three finalists — the IBM S/370, the PDP-11, and the Interdata 8/32 — for more thorough evaluation.

d. EW weighting

The architectures evaluated by the CFA committee were correlated with their performance as EW/ESM processors in Section IV.A.1. These results were then correlated to obtain the relative weights of the CFA's architectural elements from the EW/ESM performance aspect. Table IV-10 contains the relative weights of those architectural elements (as described in Section III) that most contribute to optimized EW/ESM performance.

2. Phase 1

While there are many useful parameters of a computer architecture that can be determined directly from the principles of operation manual, the only method known to be a realistic, practical test of the quality of a computer architecture is to evaluate its performance against a set of benchmarks, or test programs [21]. In the previous sections, absolute and quantitative criteria were presented that the CFA committee felt provided some indication of the quality of the candidate computer architectures. It is important to emphasize, however, that throughout the discussion of these criteria it was understood that a benchmarking phase would be needed and that many of the quantitative criteria were being used to help construct a reasonable "prefilter" that would help to reduce the number of candidate computer architectures from the original nine to a final set of three or four. As described in the previous section, this initial screening in fact reduced the set of candidate computer architectures to three: the IBM S/370, the PDP-11, and the Interdata 8/32.

This section presents the evaluation of the Computer Family Architecture (CFA) candidate architectures via a set of test programs. The measures used to rank the computer architectures were S, the size of the test program, and M and R, two measures designed to estimate the principal components contributing to the nominal execution speed of the architecture. Descriptions of the twelve test programs and definitions of the S, M, and R measures are included here. The statistical design of the assignment of test programs to programmers is also discussed. Each program was coded from two to four times on each machine to minimize the uncertanty due to programmer variability. The final results show that for all three measures (S, M, and R) the Interdata 8/32 is the superior architecture, followed closely by the PDP-11, and the IBM S/370 trailed by a significant margin.

a. Benchmark test programs

The concept of writing benchmarks or test programs is not a new idea in computer performance evaluation and is generally considered the best test of a computer system. For the purpose of the CFA committee, a *test program* was defined to be a relatively small program (100 to

Table IV-12 — Phase 1 Individual S Measure†

Test Program	Computer Architecture		
	IBM S/370	PDP-11	Interdata 8/32
A. Priority I/O Kernel	216[3]	048[4]	026[12]
	742[14]	32[14]	26[17]
	286[12]	032[12]	028[14]
B. FIFO Kernel I/O	372[2]	133[2]	144[2]
	465[13]	124[3]	142[4]
	308[17]	246[13]	098[13]
C. I/O Device Handler	192[1]	132[1]	176[1]
	252[17]	216[17]	241[17]
D. Large FFT	454[11]	766[11]	550[11]
	454[9]*	766[9]*	402[9]
			402[17]A‡
E. Character Search	104[1]	088[1]	120[1]
	092[4]	136[11]	144[3]
	154[11]	090[17]	168[11]
F. Bit Test, Set, Reset	144[9]	068[3]	082[4]
	122[12]	078[9]	090[9]
	116[17]	086[12]	098[11]A
			098[12]
G. Runge-Kutta Integration	202[2]	184[2]	166[12]
	238[17]	172[3]	158[4]
		248[17]	232[11]A
			190[17]
H. Linked List Insertion	144[4]	162[13]	148[3]
	228[13]	182[14]	198[13]
	176[14]	194[17]	164[14]
I. Quicksort	340[6]	940[6]	426[6]
	407[5]	1534[5]	524[5]
J. ASCII to Floating-Point	256[4]	164[5]	206[3]
	441[5]	208[7]	238[5]
	241[7]	172[17]	204[7]
K. Boolean Matrix	224[3]	174[4]	156[17]
	267[6]	232[6]	130[6]
	284[8]	284[8]	180[8]
L. Virtual Memory Exchange	292[3]	254[4]	328[17]
	382[7]	250[7]	310[7]
	414[8]	378[8]	334[8]

*Incomplete.

†From Ref. 21.

‡A = Auxiliary data.

Table IV-13 — Phase 1 Individual M Measure†

Test Program	Computer Architecture		
	IBM S/370	PDP-11	Interdata 8/32
A. Priority I/O Kernel	000212[3]	000028[4]	000028[12]
	000354[12]	000024[12]	000032[14]
	000522[14]	000024[14]	
B. FIFO I/O Kernel	000424[2]	000208[2]	000192[2]
	000920[13]	000188[3]	000226[4]
	000434[17]	000296[13]	000114[13]
C. I/O Device Handler	000328[1]	000309[1]	000426[1]
	000304[17]	000290[17]	000279[17]
D. Large FFT	010810[11]	014746[11]	010886[11]
	010810[9]*	014746[9]*	008560[9]*
			08560[17]A
E. Character Search	000854[1]	000730[1]	000958[1]
	000940[4]	000770[11]	001044[3]
	001724[11]	000520[17]	001021[11]
F. Bit Test, Set, Reset	000378[9]	000162[3]	000222[4]
	000358[12]	000178[9]	000176[9]
	000238[17]	000152[12]	000296[11]A
			000276[12]
G. Runge-Kutta Integration	141074[2]	102662[2]	100062[2]
	228056[17]	094960[3]	100042[4]
		176960[17]	117984[11]A
			138414[17]
H. Linked List Insertion	000228[4]	000204[13]	000224[3]
	000304[13]	000218[14]	000260[13]
	000264[14]	000240[17]	000238[14]
I. Quicksort	001024[5]	014960[5]	002968[5]
	001008[6]	002756[6]	001732[6]
J. ASCII to Float-Point	000241[4]	000292[5]	000363[3]
	000437[5]	000275[7]	000423[5]
	000433[7]	000283[17]	000334[7]
K. Boolean Matrix	000832[3]	000582[4]	000384[6]
	000909[6]	000776[6]	000566[8]
	000896[8]	000932[8]	000640[17]
L. Virtual Memory Exchange	000532[3]	000541[4]	000721[7]
	000532[7]	000566[7]	001058[8]
	000645[8]	000945[8]	000780[17]

*Incomplete.

†From Ref. 21.

‡A = Auxiliary data.

Table IV-14 — Phase 1 Individual R Measure†

Test Program	Computer Architecture		
	IBM S/370	PDP-11	Interdata 8/32
A. Priority I/O Kernel	0000947[3]	0000108[4]	000166[12]
	0002146[12]	0000106[12]	000166[17]
	0003052[14]	0000106[14]	000214[14]
B. FIFO I/O Kernel	0002222[2]	0001096[2]	000698[2]
	0004583[13]	0000810[3]	000937[4]
	0002226[17]	0001419[13]	000482[13]
C. I/O Device Handler	0001789[1]	0001480[1]	001902[1]
	0001729[17]	0001416[17]	0001391[17]
D. Large FFT	0062904[11]	0070512[11]*	060446[11]
	0062904[9]*	0070512[9]*	050045[9]
			050045[17]A‡
E. Character Search	0005603[1]	0004348[1]	005885[1]
	0005549[4]	0004326[11]	003139[3]
	0010239[11]	0003091[17]	005767[11]
F. Bit Test, Set, Reset	0001674[9]	0000832[3]	000891[9]
	0001542[12]	0000917[9]	000887[9]
	000012212[17]	000u801[12]	001167[12]
			001281[11]A
G. Runge-Kutta Integration	0845966[2]	724372[2]	696085[2]
	1203952[17]	0665529[3]	696049[4]
		1012727[17]	777846[11]A
			874923[17]
H. Linked List Insertion	0000950[4]	0001025[13]	000834[3]
	0001741[13]	0001087[14]	0001049[13]
	0001137[14]	0001210[17]	000965[14]
I. Quicksort	0007618[]	0074278[5]	013315[5]
	0007540[6]	0015205[6]	009609[6]
J. ASCII to Floating-Point	0001330[4]	0001726[5]	002100[3]
	0002578[5]	0001512[7]	002270[5]
	0002226[7]	0001716[17]	001897[17]
K. Boolean Matrix	0005576[3]	0003180[4]	002216[6]
	0005661[6]	0003905[6]	0003154[8]
	0005277[8]	0004446[8]	003945[17]
L. Virtual Memory Exchange	0001931[3]	0002616[4]	002539[7]
	0001934[7]	0002911[7]	004573[8]
	0002529[8]	0004226[8]	002643[17]

*Incomplete.

†From Ref. 21.

‡A = Auxiliary data.

500 machine instructions) selected as being representative of a class of programs. The CFA test program evaluation study had to address the central problems facing conventional benchmarking studies:

- How is a representative set of test programs selected?

- Given limited manpower, how are programmers assigned to writing test programs in order to maximize the information that can be gained?

Time is the natural measure of how quickly a test program can be executed. However, a computer architecture does not specify the execution time of any instruction and so an alternative to time must be chosen as a measure of execution speed.

The rationale of the test programs is explained below, together with the particular architectural features to be tested:

- I/O kernel, four priority levels, requires the processor to field interrupts from four devices, each of which has its own priority level. While one device is being processed, interrupts from higher priority devices are allowed.

- I/O kernel, FIFO processing, also fields interrupts from four devices, but without consideration of priority level. Instead, each interrupt causes a request for processing to be queued; requests are processed in first-in, first-out (FIFO) order. While a request is being processed, interrupts from other devices are allowed.

- I/O device handler processes application programs requests for I/O block transfers on a typical tape drive and returns the status of the transfer upon completion.

- Large FFT computes the fast Fourier transform of a large vector of 32-bit floating-point complex numbers. This benchmark exercises the machine's floating-point instructions, but principally tests its ability to manage a large address space. (Up to half a million bytes may be required for the vector.)

- Character search searches a long character string for the first occurrence of a potentially large argument string It exercises the ability to move through character strings sequentially.

- Bit test, set, or reset tests the initial value of a bit within a bit string, then optionally sets or resets the bit. It tests one kind of bit manipulation.

- Runge-Kutta integration numerically integrates a simple differential equation using third-order Runge-Kutta integration. It is primarily a test of floating-point arithmetic and iteration mechanisms.

- Linked list insertion inserts a new entry in a doubly linked list. It tests pointer manipulation.

- Quicksort sorts a potentially large vector of fixed-length strings using the Quicksort algorithm. Like FFT, it tests the ability to manipulate a large address space, but it also tests the ability of the machine to support recursive routines.

• ASCII to floating point converts an ASCII string to a floating-point number. It exercises character-to-numeric conversion.

• Boolean matrix transpose transposes a square, tightly packed bit matrix. It tests the ability to sequence through bit vectors by arbitrary increments.

• Virtual memory space exchange changes the virtual memory mapping context of the processor.

b. Measure of performance

Very little has been done in the past to quantify the relative (or absolute) performance of computer architectures, independent of specific implementations. Fundamentally, performance of computer is measured in units of space and time. The measures that were used by the CFA Committee to measure a computer architecture's performance on the test programs were S, M, and R, as follows:

Measure of Space

S: Number of bytes used to represent a test program.

Measures of Execution Time:

M: Number of bytes transferred between primary memory and the processor during the execution of the test program.

R: Number of bytes transferred among internal registers of the processor during execution of the test program.

All measures described in this section are in units of 8-bit bytes. A more fundamental unit of measure might be bits, but a number of annoying problems exist with respect to carry propagation and field alignment that make the measurement of S, M, and R in bits unduly complex. Fortunately, all computer architectures under consideration by this committee are based on 8-bit bytes (rather than 6-, 7-, or 9-bit bytes) and hence the byte unit of measurement can be conveniently applied to all these machines.

1. S measure. An important indication of how well an architecture is suited for an application (test program) is the amount of memory needed to represent it. We define $S_{i,j,k}$ to be the number of 8-bit bytes of memory used by programmer i to represent test program j in the machine language of architecture k. The S measure includes all instructions, indirect addresses, and temporary work areas required by the program.

The only memory requirement not included in S is the memory needed to hold the actual data structures, or parameters, specified for use by the test programs. For example, in the Fourier transform test program, S did not include the space for the actual vector of complex floating-point numbers being transformed, but it did include pointers used as indices into the vector, loop counters, booleans required by the program, and save-areas to hold the original contents of registers used in the computation.

2. M measure. The particular measure of primary-memory/central-processor transfers used by the CFA Committee is called the M measure. $M_{i,j,k}$ is the number of 8-bit bytes that must be read or written from primary memory by the processor of computer architecture k during the execution of test program j as written by programmer i.

3. R measure. The M measure does not capture all that is be known about the performance potential of an architecture. A second measure of architecture performance is defined: R—register-to-register traffic within the processor. Whereas the M measure looks at the data traffic between primary memory and the central processor, R is a measure of the data traffic internal to the central processor. The fundamental purpose for having the M and R measures was to enable the CFA committee to use M and R to measure a processor's execution rate. An additive measure, $aM + bR$, was used where the coefficients a and b can be varied to model projections of relative primary memory and processor speeds. An unfortunate but unavoidable property of the R measure is that it is very sensitive to assumptions about the processor's internal register and bus structure; in other words, the "implementation" of the processor.

C. Performance Results

In the following are actual measurements for each test program written for the CFA program. The unit of measurement for all data is (8-bit) bytes. The number in brackets following each measurement is the identifying number of the programmer who wrote and debugged the particular test program. Data followed by an A are auxilary data points. Data followed by a * were associated with programming assignments not completed in time to be used by the CFA Committee and the pseudovalues shown were used in the analysis of variance calculation. (When the actual data points became available at a later date, insertion of the real values for these programs had no significant effect on the results.) [21]

The statistical analysis of the above results was carried out in three phases. The average performance of the three architectures is given in Table IV-15. From the point of view of EW/ESM, the memory activity M is the best indicator of EW/ESM performance, since memory speed equates directly to throughput speed.

Table IV-15 — Average Performance of the Architectures on the 12 Test Programs*

Architecture	S	M	R	EW
PDP-11	1.00	0.93	0.94	2.293c
IBM S/370	1.21	1.27	1.29	2.182
Interdata 8/32	0.83	0.85	0.83	2.235

*After Ref. 21.

3. Phase 2

In much the same manner as Phase 1 of the previous section, a study was undertaken in support of the Military Computer Family (MCF) program to determine the relative efficiency of the following computer architectures: AN/UYK-7, AN/UYK-19, AN/UYK-20, AN/GYK-12,

and PDP-11. (The PDP-11 was recommended as a future standard military architecture for military applications in August 1976 by the Army/Navy Computer Family Architecture (CFA) Committee Phase I study. The other four architectures are the ones in most common use today in Army and Navy applications.) [24]

From a set of 160 test programs written by 16 different programmers, the following measures were found (lower is better):

Program Size	Execution Efficiency	
	Memory Activity	Processor Activity
(S measure)	(M measure)	(R measure)
PDP-11 (0.82)	AN/UYK-20 (0.73)	AN/UYK-20 (0.77)
AN/UYK-20 (0.89)		
AN/UYK-19 (0.93)	PDP-11 (0.88)	AN/GYK-12 (0.96)
	AN/GYK-12 (0.96)	PDP-11 (1.03)
AN/GYK-12 (1.14)		AN/UYK-7 (1.12)
AN/UYK-7 (1.30)	AN/UYK-19 (1.18)	AN/UYK-19 (1.17)
	AN/UYK-7 (1.38)	

The architectures are clustered into groups based on which gaps in performance were statistically significant at a practical level (i.e., the gaps in performance were statistically significant at the 95% confidence level). The numbers in parentheses give the average performance for an architecture in this study. For example, a machine with an S measure of 0.80 would require only 80% of the memory required by the average of these machines, while one with an S measure of 1.50 would require 50% more memory than the average.

The results of this study were correlated with the processor EW performance of the previous sections to obtain a weighted indicator of the relevance of these factors and benchmarks applied to EW/ESM.

The methodology used in this phase was based on a similar previous Phase I work for the CFA Committee in comparing alternative commercial architectures. However, several significant improvements have been made in the methodology of this second phase. Briefly, the differences are as follows:

- The set of test programs has been improved to be more uniform in size and wider in scope; the individual tests are more precisely directed at architectural features.

- The dynamic program measures have been extended to provide information on implementability over a range of hardware parallelism, as well as hardware speed.

- The processor activity measure has been completely redefined. The original R measure was found to be highly correlated with the original memory activity measure, and thus provided little additional information. It also failed to capture the inherent cost differences between simple and complex processor computations.

- The method of computing program measures has been automated.

- A superior statistical design was chosen that allowed more significant results to be extracted from the program measures.

A set of test programs was selected to test significant applications or capabilities of the architectures. Each program was described in a Program Description Language (PDL) that specified the algorithm to be used but left unspecified the exact machine level implementation of the algorithm. All test programs were designed to be writable by a test programmer in one or two pages of machine code.

Sixteen test programmers were selected to write test programs for the five MCF architectures. Each programmer was assigned two programs to be implemented on all five architectures. The assignment was done according to a statistical design that attempted to separate architecture effects from programmer and program effects. The programs coded by the test programmers were executed using a standard set of test data on an ISP simulator written for each machine. The Instruction Set Processor (ISP) simulator gathered statistics on the execution of the programs. Measures of efficiency computed from these statistics were used in an analysis of variance to determine the relative efficiency of each architecture.

a. Benchmark Test Programs

The set of test programs used in the MCF evaluations was constrained by budget limitations and the statistical use to be made of the results. Validity of the statistical results required that the programs be a representative set of the kind of operations performed by military computers. Along these lines, it was also considered important that the programs test all significant aspects of the architectures. These considerations would indicate the desirability of a large set of test programs. However, the analysis required that each program be coded frequently enough to allow significant statistical inferences to be made. Thus budgetary constraints forced a tradeoff between number of tests, length of test, and frequency of coding.

A set of 16 test programs divided into four categories was ultimately selected for the evaluation. The basis of the individual selections was twofold. First, a list of important architectural features was assembled. Features to be tested were

Interrupt handling and I/O
Executive/user interaction
Control and branching constructs
Integer arithmetic
Floating-point operations
Character and bit processing
Addressing mode flexibility
Ability to address large data structures.

Second, a set of significant tasks to be performed were considered:

Real-time processing
Handling multiple processes
Communications processing

Display processing
Fast table lookup
Packing and unpacking data
Sorting
Manipulation of list structures
Minimal difference search
Character processing.

The benchmark programs selected to measure the above tasks are described below (together with its EW/ESM relevance; higher is better):

• TTY Input Driver (0.457). This is a driver for a simple interrupt-driven device. Important characteristics are a low transfer rate (bytes per interrupt), minimal latency from interrupt signal to response, and high flexibility in the nature of the response. These characteristics preclude the use of a typical hardware channel (DMA transfer). The test is typical of a varity of slow-speed devices.

• Message Buffering and Transmission (0.684). A high-speed DMA device is used to transmit data buffers. The driver's concern is to buffer transmission requests and maintain as high a transfer rate as possible. The computer performs no processing on the data transmitted. This test exercises the channel (DMA) I/O structure of the architecture.

• Multiple Priority Interrupt Handler (0.189). Interrupts from four devices of unequal priority are directed to the appropriate device handlers. The I/O requests thereby completed are added to the executive's queue so that the appropriate actions may be taken relative to the requesting process. The test performs only the interrupt fielding and request queueing functions. The model is applicable to a variety of real time applications.

• Virtual Memory Exchange (0.028). A protected subroutine facility is provided by a pair of executive calls. The test program performs the memory space and register changes necessary to transfer control. The test measures supervisor call and context swap costs.

• Scale Vector Display (0.074). Given a display list and a scale factor, the program produces a scaled display list. The program is a test of integer manipulation and fixed-field extraction.

• Array Manipulation-LU Decomposition (0.192). Solution of simultaneous equations using standard Gaussian elimination. Floating-point operations, multiple indexing, and nested interation capabilities are tested.

• Target Tracking (0.295). Given the coordinates of an object, find the closest element to it in a given table. This tests floating-point comparison as well as the costs of performing contorted array searches.

• Digital Communications Processing (0.306). This program directs messages to various output lines depending upon their destinations. Fast search and block move capabilities are tested.

• Hash Table Search (0.492). The problem is to locate the position a key would occupy in a hash table. This involves address and integer manipulations and indexing.

• Linked List Insertion (0.437). Given a doubly linked list in ascending order, insert a new entry. The test involves pointer extraction and following.

• Presort on Large Address Space (0.243). Manipulate the elements of a very large, randomly ordered array to form a partially ordered binary tree. The array is large enough (order 1 Mbyte) to require manipulating the page (segment) address registers to access it. This is a test of the cost of randomly addressing a very large address space.

• Autocorrelate on Large Address Space (0.554). This test is complementary to test the Presort on Large Address Space. An autocorrelation is performed on an an array large enough to require manipulation of page registers. Floating-point and sequential access of large address spaces are tested.

• Character Search (0.229). A character string is scanned while looking for an occurrence of a specified string. This program tests character accessing abilities.

• Boolean Matrix Transpose (0.339). This program takes a bit matrix and reflects it about its diagonal. Ability to access and move bits is tested.

• Record Unpacking (0.124). This test program takes an array of tightly packed bit fields and a format string indicating the size of each field and unpacks the fields into another array. The ability to do general field extraction is tested.

• Vector to Scan Line Conversion (0.208). A list of vectors is converted to an equivalent scan line display. This tests bit addressing capabilities as well as some integer manipulations.

b. Measure of performance

As in the Phase 1 study, the performance of an architecture on the test programs is measured by the efficiency of the test programs written for that particular architecture. Quantification of the concept of an efficient program allows the comparison of different architectures independently of their implementation. The measures used by the MCF evaluation are such a quantification in terms of space and time.

An efficient program is one that requires a small amount of storage and executes in a short amount of time. Three classes of measures were used to capture this concept. The S measure is a measure of the storage requirements of a program, and the M and R measures are measures of execution efficiency.

(1). *S Measure: Test Program Size.* The S measure is defined as the number of bytes of memory required by the test program. This includes locals allocated on the stack as well as own variables. For the Interrupt and Trap test programs, this also includes memory allocated to interrupt vectors used by the test program. Excluded from the S measure are the parameter block and parameters passed to the routine as well as any global data structures to which the routine has access. This was done to avoid adding a fixed overhead of significant size to each S measure.

A single exception to the parameter exclusion principle was made. Test Program 14, Record Unpack, allowed the programmer to chose a representation for the format string. Optimal packing would cause each entry in this string to occupy 6 bits. Because a trade-off decision between packing efficiency and accessing difficulty was allowed, the size of this parameter was included in the S measure for this program.

For those test programs in which multiple calls were measured, the stack usage could conceivably vary between calls. The S measure in this case is defined as the maximum of the individual S measures.

(2). *M Measure: Memory Activity.* An important parameter of a computer system is the bandwidth of its processor/memory interface. Thus a significant determinant of program execution speed is the number of bytes the program transfers to or from memory. The M measure is a measure of memory activity.

The M measure is defined as the number of bytes read or written to main memory during the execution of the test program. Specifically, counting begins at the first instruction of the routine and ends when a return is exectuted. No activity of the calling routine is counted.

Three M measures were computed. These M measures reflect differences in the width of the memory (and therefore the minimum number of bytes that can be read from a given address). They are referred to as M8, M16, and M32- corresponding to 1-, 2, and 4-byte-wide memories, respectively.

Certainly, no one would implement the 16-bit machines with 32-bit memories without making some attempt at reasonable utilization of the wider memory. Thus, two adjustments to the M32 definition for the 16-bit machines were made. First, it is assumed that all multiple-word references (double integer, floating point, etc.) were aligned on full-word boundaries. This is of course standard practice in most 32-bit machines. Second, the sequential nature of instruction fetch makes it highly desireable to have a 32-bit instruction buffer. Otherwise a sequence of 16-bit instructions would result in each instruction being fetched twice as the low and high halves of the 32-bit word were executed. This implementation was modeled by allowing instruction fetches to fetch 2 bytes, while all other memory accesses must use 4-byte words. These two adjustments define the 32-bit memory system assumed by M32 for the 16-bit machines.

(3). *R-Measure: Processor Activity.* The activity of the processor during the execution of an instruction is simply the computation of a function. Complexity theory indicates that the cost of this computation can be measured by many step-counting functions. Consideration of step-counting functions applicable to digital implementations fails to restrict significantly the range of possible cost functions. (Consider two processors, one which is bit serial, the other uses table lookup in a read-only memory (ROM). Addition is expensive in the former, while all functions are of equal cost in the latter.) It is therefore necessary to choose a cost function that represents an implementation that is reasonable, given the current state of the art. This is the approach taken in the MCF study.

The R measure for a program is defined as the sum of the R measures for each instruction executed. The R measure of an instruction is defined as the number of CPU cycles

required to execute it using a canonical CPU. As for the M measure, no driver activity was included.

Two R measures were computed. One assumed a 16-bit-wide ALU, as would be used for low-performance versions of the AN/UYK-7 and AN/GYK-12 and most versions of the PDP-11, AN/UYK-19, and AN/UYK-20. The other assumed a 32-bit ALU, as would be used for high-performance versions of the -11, -19 and -20, and most versions of the -7 and -12. These two measures are referred to as R16 and R32.

c. Performance results

The raw scores from the Phase 1 study are shown in Tables IV-16—IV-18 together with the individual architectural EW/ESM rating.

Table IV-16 — S-Measure Results (EW Relevance= 0.275)*

Machine Prog/Pgmr	PDP-11	AN/UYK-20	AN/UYK-7	AN/GYK-12	AN/UYK-19
0/7	94	164	228	148	130
0/8	88	142	236	168	156
1/5	96	162	246	164	98
1/10	118	160	304	184	120
2/6	214	734	472	264	238
2/9	202	440	428	192	232
3/4	306	160	252	138	220
3/11	254	196	272	140	196
4/3	280	242	348	312	270
4/12	240	174	256	316	258
5/1	156	150	256	212	226
5/14	200	244	420	392	288
6/2	274	298	376	360	374
6/13	258	276	436	364	370
7/0	54	68	176	116	64
7/15	96	86	136	128	122
8/0	88	102	172	152	94
8/15	120	136	180	212	114
9/2	144	178	196	256	122
9/13	156	132	204	192	132
10/1	224	202	248	244	226
10/14	230	260	348	324	264
11/3	250	292	320	352	300
11/12	338	226	352	356	360
12/4	90	116	162	580	140
12/11	86	120	160	236	128
13/6	182	206	320	308	208
13/9	230	198	368	384	246
14/5	198	170	246	264	290
14/10	348	204	302	170	294
15/7	278	256	444	292	282
15/8	326	256	512	440	402
EW Rating (Lower is better)	2.293	1.746	1.74	1.525	2.108

*After Ref. 24.

Table IV-17 — M-Measure Results (EW Relevance = 0.150)*

Word Size	M[8]					M[16]					M[32]				
Machine	PDP-11	AN/UYK-28	AN/UYK-7	AN/GYK-12	AN/UYK-19	PDP-11	AN/UYK-20	AN/UYK-7	AN/GYK-12	AN/UYK-19	PDP-11	AN/UYK-20	AN/UYK-7	AN/GYK-12	AN/UYK-19
Prog/Pgmr															
0/7	2360	1088	7610	3524	4206	2512	1164	7782	3578	4206	3554	1528	7872	3904	6020
0/8	2294	1088	7568	3097	4052	2370	1164	7700	3152	4052	3506	1530	7700	3482	5886
1/5	252	354	572	434	220	258	354	572	434	220	366	504	572	444	302
1/10	350	418	884	576	326	350	418	884	576	326	488	574	884	588	472
2/6	620	1268	1312	528	502	620	1268	1312	528	502	880	1638	1312	528	726
2/9	460	872	1340	364	876	466	878	1340	364	876	632	1250	1340	416	1274
3/4	618	496	512	188	424	620	496	512	188	424	986	874	512	220	660
3/11	666	576	611	190	574	668	576	618	190	574	1014	974	620	216	972
4/3	2598	1422	3144	1804	4010	2598	1422	3144	1804	4010	3370	1546	3144	1808	5342
4/12	2084	1064	1670	1716	2906	2084	1064	1880	1716	2906	2550	1256	1880	1716	3760
5/1	794	750	1656	964	1858	794	750	1656	964	1858	1094	1068	1656	964	2250
5/14	2018	1752	5396	4660	2978	2018	1752	5396	4660	2978	2634	2198	5396	4660	3554
6/2	2644	2848	5456	3288	4472	2654	2848	5456	3288	4472	3595	3618	5456	3368	4884
6/13	2610	2370	4592	3340	3778	2610	2370	4592	3340	3778	3364	2982	4592	3380	4118
7/0	960	966	2214	1840	1754	960	966	2214	1840	1754	1428	1432	2236	1868	2202
7/15	1510	1088	2414	2114	2622	1510	1088	2414	2114	2622	1948	1530	2416	2120	4300
8/0	898	818	1492	1246	838	902	818	1492	1246	838	1216	1028	1544	1324	1098
8/15	1140	1022	2232	1708	986	1140	1022	2232	1708	986	1548	1250	2232	1708	1372
9/2	220	278	396	344	230	220	278	396	344	230	286	370	396	364	294
9/13	282	210	372	304	256	282	210	372	304	256	376	288	372	304	344
10/1	2500	1376	1468	1992	2542	2500	1376	1468	1992	2542	3202	1660	1468	1992	2986
10/14	3042	1948	3864	3356	3226	3042	1948	3864	3356	3226	3696	2144	3864	3364	3694
11/3	11838	9100	7000	10196	14040	11838	9100	7000	10196	14040	14819	11094	7000	10196	16220
11/12	6880	4170	5892	5216	9832	6880	4170	5892	5216	9832	8998	5420	5892	5216	11606
12/4	762	992	2529	1636	2366	832	1062	2808	1660	2364	1098	1340	2808	1732	3000
12/11	842	1370	3041	2668	2630	912	1440	3320	2668	2630	1218	1868	3320	2872	3602
13/6	1490	882	2492	1896	1936	1540	882	2492	1896	1936	1918	1044	2492	1996	2436
13/9	700	540	1520	1066	916	700	540	1664	1066	916	926	652	1664	1144	1242
14/5	769	516	950	736	1312	782	522	968	736	1312	1008	626	968	760	1796
14/10	2082	660	846	676	2488	2088	660	864	676	2488	2370	784	864	700	2936
15/7	4404	3666	5818	4962	7702	4404	3666	5818	4962	7702	5732	4644	5904	5172	9458
15/8	7046	5004	8442	5686	8236	7046	5004	8442	5686	8236	8870	6234	8532	6100	10536
EW Rating (Lower is better)	2.293	1.746	1.174	1.525	2.108	2.293	1.746	1.174	1.525	2.108	2.293	1.746	1.174	1.525	2.108

*After Ref. 24.

Table IV-18 — R-Measure Results (EW Relevance = 0.452)*

Wordsize Machine Prog/Pgmr	R[16]					R[32]				
	PDP-11	AN/UYK-20	AN/UYK-7	AN/GYK-12	AN/UYK-19	PDP-11	AN/UYK-20	AN/UYK-7	AN/GYK-12	AN/UYK-19
0/7	2631	1009	6049	2281	4450	2631	1009	4238	1994	4450
0/8	1932	945	6282	2060	4360	1932	945	4255	1729	4360
1/5	237	249	311	313	246	237	249	274	237	246
1/10	322	292	550	385	367	322	292	401	290	367
2/6	547	1031	819	366	485	547	1031	638	280	485
2/9	496	694	800	252	960	496	694	603	221	960
3/4	559	130	611	168	328	543	130	509	125	328
3/11	750	179	659	171	334	734	179	538	130	334
4/3	4008	3261	7939	4055	5323	4008	3261	4509	3170	5323
4/12	4131	3195	3309	7109	4779	4131	3195	1983	3964	4779
5/1	2262	2069	2541	2346	2037	1558	1491	1606	1349	1795
5/14	4472	4207	7539	7834	4602	3766	3629	4692	4602	4360
6/2	4991	4526	4893	4229	4632	4049	4064	3401	2927	4257
6/13	5172	4034	4526	4347	4300	4102	3784	3116	2961	3897
7/0	1488	1203	1364	1651	2454	1488	1203	1175	1362	2454
7/15	2001	1314	1807	1974	1430	2001	1314	1476	1397	1430
8/0	1402	1086	1549	1326	1193	1402	1086	1059	1001	1193
8/15	1867	1688	3146	2331	1728	1867	1688	2008	1734	1728
9/2	212	238	223	291	308	212	238	205	210	308
9/13	290	207	225	244	311	290	207	205	175	311
10/1	2981	1956	1309	2254	2746	2981	1956	890	1606	2746
10/14	3893	2456	2074	3336	3371	3893	2456	1647	2264	3371
11/3	17995	14443	10095	15152	14367	16176	12848	6324	9758	14202
11/12	9389	7750	7868	8933	7723	7544	6155	4977	5511	7558
12/4	1169	1175	1647	1789	3051	1169	1175	1209	1263	3051
12/11	1149	1083	1706	2943	2843	1149	1083	1413	2153	2843
13/6	2277	1903	3882	3781	3200	2277	1903	2404	2233	3200
13/9	733	836	1084	1130	1059	733	836	815	838	1059
14/5	862	781	764	917	1640	862	781	558	657	1640
14/10	3685	886	738	851	3625	3685	886	534	566	3625
15/7	8308	6696	11827	11634	11751	8308	6512	7287	6939	11751
15/8	11397	6660	8207	6174	10173	11397	6476	5791	4414	10173

*After Ref. 24.

These results are summarized in Table IV-19 and at the beginning of this section in Table IV-1. It should be noted that the best score for general military application was not necessarily the best for EW/ESM application for various reasons as have been discussed in previous sections, this section, and subsequent sections. Further, it should be noted that certain of the MCF benchmark test programs displayed high EW relevance for certain measures (0.983 for Autocorrelation, S-measure), while certain other benchmarks displayed almost no EW relevance (0.000 for Virtual Memory Exchange, M-measure), indicating definite EW/ESM applicability.

Table IV-19 — Phase 2 Results
(Lower is better)

Machine	EW Rating	Military Application		
		S-measure	M-measure	R-measure
AN/UYK-7	1.174	1.30	1.38	1.12
AN/GYK-12	1.525	1.14	0.96	0.96
AN/UYK-20	1.746	0.89	0.73	0.77
AN/UYK-19	2.108	0.93	1.18	1.17
PDP-11	2.293	0.82	0.88	1.03

V. SOFTWARE

Also necessary to the specification of a computer processor are the software requirements, especially the software tools. Subsequently, it will be shown in the cost section that the programming cost will increase the total processor-system cost depending upon the software "tools" that are available. The cost of the hardware architecture will be insignificant if there is not a good set of software tools available to support the programming. The factor Tool Availability Index (TAI) is introduced to represent the architectural software tools. The TAI will be subsequently shown to be a direct function of the architectural inventory investment in dollars. The cost of programming will be shown to be a direct function of the TAI. The Military Computer Family (MCF) program carried out a study of software tools for five common military architectures under Harold S. Stone of the University of Massachusetts. These tools, as they appeared in Stone's report, were examined for applicability to EW/ESM through the process of architectural performance in the EW/ESM index of Section IV. This section contains a description of the tool, its dollar value (to construct), and the EW/ESM applicability.

A. Software Tool Description

This section contains a description of the existing software base for the military computers AN/UKY-7, AN/UYK-20, AN/GYK-12, and AN/UYK-19. For reference, the DEC/PDP-11 is included for comparison and linkage to the other sections of this report. The software study treated the set of software tools selected by a committee of Army/Navy representatives during the selection of a candidate architecture for the standard military computer family [23].

The method used to assess the software base for each computer was to visit the manufacturer of each computer system and principal users, questioning the individuals about the available software. Where questions arose, the questions were answered by examining reference documentation. A tool was deemed to be available for a specific architecture if:

(1) the tool was a genuine released item supported by user documentation,

(2) the tool had to be available for use as of January 1, 1977, and not merely nearing release, and

(3) the tool had to satisfy the characteristics for that tool as described in Ref. 7.

In evaluating the software base, the study was enlarged to include software systems that are not self-hosted. This gives rise to some interpretation as to what tools should be counted in the software base of a particular architecture and what tools should be treated as missing. In several instances, the existence of a software tool for one system makes that tool available to all architectures, since the tool itself need not be self-hosted. For example, one tool in the base is a General Purpose Systems Simulator, of which GPSS for the IBM System/370 is but one example. Since this tool is not likely to be deployed in tactical systems but is more likely to be used at software development centers, it does not make sense to insist that it run on a particular computer family provided that some computer that supports the tool is available at each development center. In the case of the simulator, there is a GPSS system or equivalent for every major commercial computer system, and it is extremely likely that at least one commercial computer will be accessible by support software staff. Consequently, the availability of GPSS counts as a simulator in the software base of each candidate architecture.

GPSS was selected for this example, since its function is completely independent of the architecture of the computers involved. Other tools that share this characteristic are indicated. Examples of software tools that do not have this characteristic are compilers, assemblers, and operating systems, since these are dependent in some essential way on the target computer architecture.

The inclusion of nonself-hosted software in this study has been done by considering each tool in two forms—self-hosted and nonself-hosted. The listings are compiled separately. Some tools simply cannot be used except in self-hosted form, so these do not appear in the tabulation of nonself-hosted software. Among these tools are operating systems and language-independent monitors, since both of these depend on the execution of the tool on the host architecture in real time. Other tools omitted from the nonself-hosted list have been excluded for similar reasons. The software base for the AN/AYK-14 is essentially that of the AN/UYK-20 since the AYK-14 is upward compatible from the AN/UYK-20, and the group in charge of the AN/UYK-20 software is currently making the few modifications required to move the entire base to the AN/AYK-14. Although the AN/AYK-14 was not included in this study, very little error is introduced by treating its base equal to the AN/UYK-20 base.

1. *Self-hosted software.*

Table V-1 gives the pertinent results of the software base study for self-hosted software. Details of the table are explained in this section. We list only the tools that exist for at least one architecture.

Tool 1.a. Data Base Design Aid. This tool is used for the construction of data base systems. For the AN/UYK-21 the appropriate tool is the SCHEMA Compiler, which is part of the IDMS-11 data base management system for the PDP-11 and is available from Digital Equipment Corporation.

Table V-1 — Software Support Tools

Tool	Performance General/EW (Lower is better)	AN/UYK-7 1.30/0.62	AN/UYK-19 0.93/1.56	AN/UYK-20 0.89/0.94	AN/GYK-12 1.14/0.93	PDP-11 0.82/1.59	IBM 0.99/1.34	Interdata 0.68/1.42
Computer System Simulator	0.99/1.34						0.99/1.34	
Data Base Design Aid	0.91/1.47					0.82/1.59	0.99/1.34	
Test Case Design Advisors	N/A							
FORTRAN								
CMS-2								
TACPOL								
DOD-1								
Test Case Instrumenters & Analyzers	0.96/1.12							
FORTRAN	0.86/1.48		0.93/1.56			0.82/1.59	0.99/1.34	0.68/1.42
CMS-2	0.89/0.94			0.89/0.94				
TACPOL	1.14/0.93				1.14/0.93			
DOD-1								
Compilers & Cross-compilers	1.03/1.06							
FORTRAN	0.86/1.48		0.93/1.56			0.82/1.59	0.99/1.34	0.68/1.42
CMS-2	1.10/0.78	1.30/0.62		0.89/0.94				
TACPOL	1.14/0.93				1.14/0.93			
Assembler	0.96/1.20	1.30/0.62	0.93/1.56	0.89/0.94	1.14/0.93	0.82/1.59	0.99/1.34	0.68/1.42
Macro Assembler	0.96/1.20	1.30/0.62	0.93/1.56	0.89/0.94	1.14/0.93	0.82/1.59	0.99/1.34	0.68/1.42
Basic Linker	0.96/1.20	1.30/0.62	0.93/1.56	0.89/0.94	1.14/0.93	0.82/1.59	0.98/1.34	0.68/1.42
Simple Overlay Linker	0.85/1.48		0.93/1.56			0.82/1.59	0.98/1.34	0.68/1.42
Extended Overlay Linker	0.91/1.47					0.82/1.59	0.99/1.34	
Interactive Debugging Aids	1.00/1.31							
Assembler	1.01/1.28	1.30/0.62	0.93/1.56			0.82/1.59	0.99/6.34	
FORTRAN	0.99/1.34						0.99/1.34	
TACPOL								
DOD-1								
CMS-2								
Noninteractive Debugging Aids	1.12/1.01							
Assembler								
FORTRAN	0.91/1.47					0.82/1.59	0.99/1.34	
CMS-2	1.30/0.62	1.30/0.62						
TACPOL	1.14/0.93				1.14/0.93			
DOD-1								

Table V-1 (continued) — Software Support Tools

Tool	Performance General/EW (Lower is better)	AN/UYK-7 1.30/0.62	AN/UYK-19 0.93/1.56	AN/UYK-20 0.89/0.94	AN/GYK-12 1.14/0.93	PDP-11 0.82/1.59	IBM 0.99/1.34	Interdata 0.68/1.42
Language Independent Monitor	1.10/0.78	1.30/0.62		0.89/0.94				
Language Dependent Monitor	0.97/1.24							
Assembler	0.94/1.14			0.89/0.94			0.99/1.34	
FORTRAN	0.99/1.34						0.99/1.34	
CMS-2								
TACPOL								
DOD-1								
Reformatters	0.95/1.24							
FORTRAN	0.95/1.24	1.30/0.62				0.82/1.59	0.99/1.34	0.68/1.42
CMS-2								
TACPOL								
DOD-1								
Standards Enforcers	N/A							
FORTRAN								
CMS-2								
TACPOL								
DOD-1								
Test Data Auditor	1.01/1.16	1.30/0.62	0.93/1.56	0.89/0.94	1.14/0.93	0.82/1.59	0.99/1.34	
Test Data Generator	1.01/1.16	1.30/0.62	0.93/1.56	0.89/0.94	1.14/0.93	0.82/1.59	0.99/1.34	
Integrated Library	0.99/1.21	1.30/0.62	0.93/1.56	0.89/0.94		0.82/1.59	0.99/1.34	
Automatic Software Production & Test	N/A							
Text Processing System	1.01/1.16	1.30/0.62	0.93/1.56	0.89/0.94	1.14/0.93	0.82/1.59	0.99/1.34	
Interactive Source Editor	0.96/1.20	1.30/0.62	0.93/1.56	0.89/0.94	1.14/0.93	0.82/1.59	0.99/1.34	0.68/1.42
Batch Source Editor	0.96/1.20	1.30/0.62	0.93/1.56	0.89/0.94	1.14/0.93	0.82/1.59	0.99/1.34	0.68/1.42
Data Base Management System	1.01/1.28	1.30/0.62	0.93/1.56			0.82/1.59	0.99/1.34	
Real-Time + Time-Sharing OS	0.95/1.24	1.30/0.62				0.82/1.59	0.99/1.34	0.68/1.42
Real-Time + Time-Sharing + VMM OS	0.99/1.34						0.99/1.34	
Time-Sharing + Multi-Processing + VMM OS	0.99/1.34						0.99/1.34	
Instruction simulator	0.99/1.18	1.30/0.62	0.93/1.56	0.89/0.94		0.82/1.59		
General Purpose System Simulator	1.01/1.16	1.30/0.62	0.93/1.56	0.89/0.94	1.14/0.93	0.82/1.59	0.99/1.34	

Tool 1.b. Test Instruments and Analyzers. This tool allows a programmer to instrument his source program so as to obtain counts of the number of times designated portions of his program are executed. He can also discover which portions have not been executed at all. Since the tool is specific to particular source languages, and since it may itself be written in a machine-independent source language, it may be available for all architectures if it is available for one. Indeed, this is the case for a FORTRAN analyzer. Either RXVP-1 or TAP may be used; both are commercially available. They are written in FORTRAN and can be moved to any computer system with a FORTRAN compiler. Thus they count in the base of all architectures except the AN/GYK-12, since it does not have a FORTRAN compiler in its base.

The AN/GYK-12 has a TACPOL compiler with the test instrumentation built into the language, so that it gets credit for this tool in the TACPOL base.

Tool 1.c.1 Basic Assembler

All architectures have assemblers available. In several cases there is more than one assembler. Since macro assemblers satisfy the basic assembler need, and all architectures have macro assemblers, we list the source of the tool under the macro assembler heading in the next paragraph.

Tool 1.c.2. Macro Assembler

The macro assemblers available for the several computers are the following:

AN/UYK-7 MACRO/32 (from Univac)
AN/UYK-20 Level 2 Assembler and MACRO/20 (from Univac and NAVSEC)
AN/GYK-12 L3050 Macro Assembler (from Litton)
AN/UYK-19 Macro Assemble (from ROLM)
AN/UYK-21 MACRO/11 (from DEC)

Tool 1.d.1 FORTRAN Compiler

All architectures except the AN/GYK-12 have a FORTRAN compiler available. Some have more than one. Univac and NAVSEC are sources for the AN/UYK-7 and AN/UYK-20 compilers while ROLM and DEC supply compilers for the AN/UYK-19 and AN/UYK-21, respectively.

Tool 1.d.2. COBOL Compiler

The two architectures compatible with commercial computers have COBOL compilers, while the purely military architectures do not. The COBOL for the AN/UYK-19 is called IPI COBOL, available from the Florida-based company IPI. DEC supplies the COBOL for the AN/UYK-21.

Tool 1.d.3 CMS-2 Compiler

For the purposes of this study we do not distinguish among the several different levels of CMS-2. For the architectures considered here, if a CMS-2 compiler exists, then the study

found that it exists for at least two levels of the language definition. Univac and NAVSEC can supply the CMS-2 compilers for the AN/UYK-7 and AN/UYK-20 architectures. Command Control Communications Corporation of San Pedro, California, supplies the CMS-2 for the AN/UYK-19.

Tool 1.d.4 JOVIAL Compiler

Univac has written a JOVIAL for the UYK-7 and can supply this as a product.

Tool 1.d.5 TACPOL Compiler

Litton's TACPOL compiler for the AN/GYK-12 is called TACPOL-B and is available from Litton.

Tool 1.e.1 Basic Linker

Architectures that had simple overlay linkers were credited with having basic linkers as well. The linking capability exists for all architectures, at least at the basic level, and in several instances at the simple overlay level. The linkers credited to each architecture are as follows:

AN/UYK-7 Linker (from Univac)
AN/UYK-20 Linking loader (from Univac and NAVSEC)
AN/GYK-12 Embedded in PSS and TOS operating systems (from Litton)
AN/UYK-19 RLDR (from Data General and ROLM)
AN/UYK-21 LINK-11 (from DEC)

Tool 1.e.2. Simple Overlay Linker

This tool performs basic linking of modules and has the capability of constructing overlay trees so that overlay calls from module to module can be implemented. The programs and their respective sources are given as follows:

AN/UYK-19 RLDR (from Data General and ROLM)
AN/UYK-21 RSX TASK BUILDER (from DEC)

Tool 1.e.3. Extended Overlay Linker

This tool has all of the capabilities of a simple overlay linker plus the ability to manage memory dynamically so as to take advantage of available memory. There is some concern as to the applicability of this tool in a system with virtual memory, since the virtual memory achieves the same function in a different way. Only the AN/UYK-21 has such a linker, namely in the RSX Task Builder available from DEC. The tool known as DMR for Dynamic Memory Restructurer, available for the AN/UYK-7, has some of the capability of an extended overlay linker in that it can restructure memory assignments dynamically in a system in which one or more memory modules have failed. However, its user interface does not appear to be adequate for the intended purposes of the extended overlay linker, since the main function of DMR is reconfiguration for reliability purposes.

Tool 1.f.1. Interactive Debugger for Assembler

This tool is deemed to be similar to debugging systems similar to On-Line Debug Trace (OBT) and Dynamic Debugging Technique (DDT) for the PDP-10 and PDP-11 computer systems.

The debugging aids given in Table V-1 are QED for the AN/UYK-7 (from Litton), DEBUG for the AN/UYK-19 (from ROLM) and DDT for AN/UYK-21 (from Carnegie-Mellon University and DEC).

Tool 1.f.2. Symbolic Debugger for COBOL

This tool is the COBOL equivalent of the previous tool in that one has to be able to debug interactively at the source language level with the aid of this tool. This presupposes that COBOL exists on the architecture. The debugging aids are built-in features of the two architectures that have COBOL compilers. For the AN/UYK-19, the program is available from Data General and for the AN/UYK-21 it is available from DEC.

Tool 1.f.3. Noninteractive Symbolic Debugger for FORTRAN

Of the architectures for which FORTRAN is available only the AN/UYK-21 architecture has a FORTRAN with built-in symbolic debugging aids for batch (noninteractive) debugging of FORTRAN. The source of the software is DEC and it is contained in PDP-11 FORTRAN.

Tool 1.f.4. Noninteractive Symbolic Debugger for COBOL

This is the same situation as for the FORTRAN batch debugging aid. Only the AN/UYK-21 architecture has such software available, and it is part of the COBOL package for the PDP-11 available from DEC.

Tool 1.f.5 Noninteractive Symbolic Debugger for CMS-2

The CMS-2 compiler available from Univac for the AN/UYK-7 has debugging facilities for noninteractive debugging at a symbolic level.

Tool 1.f.6. Noninteractive Symbolic Debugger for TACPOL

The TACPOL language as implemented by Litton for the AN/GYK-12 contains debugging directives that satisfy this requirement.

Tool 1.g. Integrated Library

This tool is a comprehensive library change control system. It exists for the UYK-21 architecture in the form of the Modification Request Control System and Source Code Control System that are part of the Programmer's Work Bench System available from DEC for the PDP-11. For the AN/GYK-12, Litton has developed a tool known as SPS Librarian. Other architectures apparently do not have software that meets the specifications for this tool.

Tool 1.h.1. Language-Dependent Monitor for Assembler

There exists a tool known as STV, for System Test Vehicle, available from Univac and NAVSEC that appears to meet the tool specifications. This runs on the AN/UYK-20.

Tool 1.h.2. Language-Independent Monitor

The STV for AN/UYK-20 satisfies this tool requirement and is available from Univac and NAVSEC. A monitor for the AN/UKY-7 is available from Litton as part of Interim Tactical Amphibious Warfare Data System (ITAWDS) software system written to support the LHA (Landingship Heavy Amphibious) project.

Tool 1.i. Reformatters

All architectures that have FORTRAN compilers are deemed to have this tool since it is available as a FORTRAN source to operate on FORTRAN as a preprocessing step. The tool is called IFTRAN-2 FORTRAN Preprocessor for Structure Programming. Since the AN/GYK-12 does not have a FORTRAN compiler, it is missing this tool. However, if a FORTRAN compiler becomes available for the AN/GYK-12, then this tool becomes available automatically. Thus the absence of this tool does not penalize the AN/GYK-12's software base.

Tool 1.j. Data Base Management System

The tool known as ITAWDS mentioned above for language-independent monitors is a comprehensive software system that contains within it a data base management system. It is available from Litton. The two architectures that have commercial counterparts also have data base management systems. For the AN/UYK-19, the system is known as MIDAS, and is available from Boeing Computer Services. For the AN/UYK-21, the tool is IDMS-11, available from DEC for the PDP-11.

Tool 1.k. Text-Processing System

The text-processing system function is to prepare final copy of publishable materials, and it has many built-in features that assist this process. The text processor listed for the AN/UYK-7 is available from Fleet Combat Direct Systems Support Activity (FCDSSA). The tool listed for the AN/UYK-21 is TYPESET-11 and is available from DEC for the PDP-11.

Tool 1.l. Interactive Source Language Editor

The editor of the AN/UYK-19 architecture is known as SPEED, and is available from ROLM. The AN/UYK-7 editor is a subsystem of SHARE 7 and is available from Univac. Any one of a number of editors for the AN/UYK-21 architecture meet this criterion. Among them is the editor that runs under IAS available from DEC for the PDP-11.

Tool 1.m. Batch Source Language Editor

All architectures have batch editors available according to the list below:

AN/UYK-7 Subsystem of Share 7 (from Univac)
AN/UYK-20 Subsystem of Level 2 Librarian system (from Univac)
AN/UYK-19 BEDIT (from ROLM)
AN/GYK-12 Subsystem of SPS Librarian (from Litton)
AN/UYK-3 SLIPER (from DEC).

Tool 1.n. Real-Time Operating System plus Time-Sharing

The AN/GYK-12 version of this tool is the ITAWDS system available from Litton. The AN/UYK-21 software system is called IAS and is available from DEC.

Tool 1.o Time-Sharing Plus Virtual Machine Monitor

Of the five architectures only the AN/UYK-21 was given credit for being virtualizable in the sense of being able to support a virtual machine monitor as of January 1, 1976. The architectures have not evolved since that time to the point where any others have been shown to be virtualizable. However, not even the AN/UYK-21 was given credit for this tool, since the only instance of a virtual machine monitor for the AN/UYK-21 exists in a research environment and cannot said to be a releasable piece of software. For similar reasons, no architecture was credited with Tool 3.14, which is a superset of this tool.

2. *Nonself-Hosted Software*

Software that can be hosted on architectures other than the candidate architectures must necessarily not depend on real-time responses of the candidate architectures. This eliminates operating systems, performance monitors, and data base management systems from the list of tools that can be self-hosted. Similar reasoning reduces the list of possible nonself-hosted tools to the list given in Table V-1. Since some tools fall in the category of those that are available for all architectures if available for one, they are credited to all architectures. The text in this section explains all such cases.

Tool 2.a General-Purpose System Simulator

This program exists on the IBM 370 and Univac 1100 series computers, among others. It is probably not necessary to mount a special development to create a running version on a military computer system.

Tool 2.b Cross-Assembler

All architectures have at least one cross-assembler that executes on a foreign architecture, producing object code for the native architecture. In some cases the cross-assembler is written in a machine-independent language such as FORTRAN so that it can run on any one of several different computers. The list of cross-assemblers available is as follows:

AN/UYK-7 Available for Univac 1100 series computers from Univac.

AN/UYK-20 Runs on six different computers since it is written in compatible FORTRAN (from NAVSEC).

AN/UYK-19 Available for IBM 370, Univac 1100, and CDC 6000 series computers from Computer Associates and First Data

AN/GYK-12 Available for IBM 370

All assemblers that run on foreign hardware are macro assemblers.

Tool 2.c.1. FORTRAN Compiler

The AN/UYK-20 architecture has a FORTRAN compiler written in FORTRAN that has successfully executed on CDC 6000, Univac 1100, and IBM 370 series computers.

The AN/UYK-21 has a FORTRAN cross-compiler that runs on the GE 6000 computers and is available from GE.

Tool 2.c.2. COBOL Compiler

A COBOL compiler that runs on the IBM 370 and generates code for the AN/UYK-19 has been written and is in operation by the Navy.

Tool 2.c.3. CMS-2 Compiler

The CMS-2 compiler for the AN/UYK-19 mentioned above is written in FORTRAN and runs on several different computers. Similarly, the CMS-2 compiler for the AN/UYK-20 is written in FORTRAN and runs on several different computers. There is an AN/UYK-7 CMS-2 compiler that runs on Univac 1100 computer systems, and the nonself-hosting methodology is the normal mode of operation with CMS-2 program development.

Tool 2.c.4. JOVIAL Compiler

JOVIAL is written in itself and can run on several different computers while producing object code for a specific computer. Since there is self-hosted JOVIAL compiler for the AN/UYK-7, this compiler can be and has been run successfully on other computers that have JOVIAL.

Tool 2.c.5. TACPOL

TACPOL-A is the TACPOL compiler for the AN/GYK-12 that executes on the IBM 370 series computers.

Tool 2.d. Instruction Simulators

All but the AN/UYK-21 have reported nonself-hosted instruction simulators. We suspect that there are such simulators for the AN/UYK-21 as well, and are currently investigating this possibility. The ISP compiler at Carnegie-Mellon University does simulate the PDP-11 correctly at instruction level, but it is not a piece of software with extensive outside release and use, so it is not counted here. A summary of the instruction-level simulators appears below:

AN/UYK-7 Runs on the Univac 1100 series computers (from Univac)

AN/UYK-20 FORTRAN-based program for AN/UYK-20 runs everywhere (from Univac)

AN/GYK-12 Available for IBM 370 (from Litton)

AN/UYK-19 Available for IBM 370 (from ROLM)

AN/UKY-21 Available for IBM 370 (from First Data and Computer Associates).

3. *Summary*

Some caution concerning the accuracy of the data and its validity for use in cost models should be observed. There may be some tools that are missed. Several sources, where possible, were sought for an architecture with good agreement. It is rather unlikely that anything significant was missed, and highly probable that all data given are correct as of this date (1977).

The table contains a single bit of information about each item, that is, whether it exists in releasable form or not. Two different software systems that ostensibly perform the same function may have vastly different characteristics, utility to the user, and procurement costs.

B. Software Tool Evaluation

The efficiency with which a programmer programs a particular architecture depends directly on the Tool Availability Index (TAI) mentioned previously. The TAI is difficult to evaluate directly. The elements of TAI, as described in the previous section, were cataloged by participants of the CFA/MCF programs for some prime military architectures. These TAI elements were quantified in Table V-1, by using the performance of these architectures as previously discussed. The criterion of performance was the S-measure of program efficiency for those machines as applied to the particular architectures in general military usage. A lower index indicated greater utility in the programming of that machine. Where a number of machines had that TAI element available, a mean was used to quantify the particular tool [25].

In terms of EW application, the S-measure of Benchmark programs with particularly high EW correlations were used and depicted as the second entry under the representative architecture of Table V-1. The results of these entries were ordered from most contributing (lowest numerical index) to least contributing in the two lists that follow.

Priority List of Software Tools (General Military)

Performance (Lower is Better)	Tool
0.85	Simple Overlay Linker
0.91	Extended Overlay Linker
0.91	Data Base Design Aid
0.95	Real Time + Time Sharing Operational System
0.95	Reformatters

0.96	Test Case Instrumenters + Analyzers
0.96	Batch Source Editor
0.96	Assembler
0.96	Interactive Source Editor
0.96	Macro Assembler
0.96	Basic Linker
0.97	Language-Independent Monitor
0.99	Computer System Simulator
0.99	Integrated Library
0.99	Real-Time + Time Sharing + Virtual Machine Monitor Operational System
0.99	Time-Sharing + Multi-processing + Virtual Machine Monitor Operational System
0.99	Instruction Simulator
1.00	Interactive Debugging Aids
1.01	Test Data Auditor
1.01	Test Data Generator
1.01	Text Processing System
1.01	Data Base Management System
1.01	General Purpose Simulator
1.03	Compilers + Cross-Compilers
1.10	Language-Independent Monitor
1.12	Noninteractive Debugging Aids

Priority List of Software Tools (Electronic Warfare)

Performance (Lower is Better)	Tool
0.78	Language-Independent Monitor
1.01	Noninteractive Debugging Aids
1.06	Compilers + Cross-Compilers
1.12	Test Case Instrumenters + Analyzers

1.16	Test Case Auditor
1.16	Test Case Generator
1.16	Text Processing System
1.16	General Purpose System Simulator
1.18	Instruction Simulator
1.20	Assembler
1.20	Macro Assembler
1.20	Basic Linker
1.20	Interactive Source Editor
1.20	Batch Source Editor
1.21	Integrated Library
1.24	Language Dependent Monitor
1.24	Reformatters
1.24	Real Time + Time-Sharing Operational System
1.28	Data Base Management System
1.31	Interactive Debugging Aids
1.34	Computer System Simulator
1.34	Real Time + Time-Sharing + Virtual Machine Monitor Operational System
1.34	Time-Sharing + Multiprocessing + Virtual Machine Monitor Operational System
1.47	Extended Overlay Linker
1.47	Data Base Design Aid
1.48	Simple Overlay Linker

The rationale of these figures lies in the facts that a machine using certain of the tool list and displaying good programing efficiency through the S-measure, achieved that efficiency through those tool elements. It follows that those tool elements should be noted with that efficiency on the mean of those efficiencies when a number of machines are involved with the same tool element. There are fallacies in this approach in that a little-used tool may appear to be more efficient than a widely used tool that may appear on machines that are poor performers for other reasons; such as the "Simple Overlay Linker, 0.85 compared to "Compilers & Cross-Compilers, 1.05." The latter tool obtained a poorer performance rating only because it appeared on all machines; whereas, the former tool "Simple Overlay Linker," appeared only with those machines with a very efficient program performance S-measure. Obviously, the machines cannot function without certain basic tools, inefficient or not. Small differences in performance indices are not considered significant.

Table V-2 attempts to distinguish the performance of some of the various common military languages, such as FORTRAN, CMS-2, and TACPOL, for application to general military and EW. Interestingly enough, FORTRAN, 0.93, shows the most efficient performance for general military application; whereas, CMS-2, 0.78, displays the best EW performance efficiency.

Table V-2 — Software Language Efficiency

Software Tool	FORTRAN	CMS-2	TACPOL
Test Case Design Advisor	-	-	-
Test Case Instr + Analyze	0.86/1.48	0.89/0.94	1.14/0.93
Compilers + Cross-compilers	0.86/1.48	1.10/0.78	1.14/0.93
Interactive Debug Aids	0.99/1.34	-	-
Non-Interact Debug Aids	0.91/1.47	1.30/0.62	1.14/0.93
language Dependent Monitor	0.99/1.34	-	-
Reformatters	0.95/1.24	-	-
Standards Enforcers	-	-	-
Performance(GM)	0.93±.06	1.10±.21	1.14±.00
(Lower is Better)(EW)	1.39±.10	0.78±.16	0.93±.00

VI. SPECIFICATION

When the military, Navy, or EW system manager decides to procure a computer/processor, he is more likely to be procuring an existent architecture that is being adopted to the particular needs of the system, or an architecture with existing standard options. Otherwise, the procurement becomes an exceptionally expensive development project with subsequently expensive software and hardware support to keep the computer/processor operating. This is not to say that procurement cannot be competitive or innovative; however, the procurer should not attempt to specify architecture or technology. He should specify capacities, performance, physical environment, reliability, maintainability, basis of selection, weighting factors, and general software support tools. It will be the vendor's responsibility to respond with architecture, technology, and supporting software tools that he considers best fit the procurement specifications.

It is important that the procurer understand technology well enough to not overspecify his requirements, which would result in no-response or a request for waiver of specifications. In addition, it is important that the procurer understand technology well enough that he may evaluate the responses to this procurement. Of course when "he" is used in this context, "he" may refer either to an individual or a board of experts depending upon the size (value) of the procurement.

The simplest procurement is by catalogue wherein the vendor's options are fitted to the procurer's requirements. Here it is necessary to define requirements in terms of computer/processor capacity and performance. In such a case, the procurer must accept what he is getting based upon the the vendor's specification.

A. Hardware

The greatest single factor affecting processing speed is the interplay between the memory access time and the microinstruction cycle time. Techniques have been shown that can be used to increase the operating speed of a computer sytem consisting of a fast processor coupled to a slow memory.

For a computer system where the memory setup and access times are faster than a central processor's minimum cycle time, the instruction execution time depends entirely on the central processor. Therefore, the evaluation of the computer, according to the principles laid down, is equivalent to the evaluation of the central processor.

For a computer system where the memory access time is slightly greater than the central processor's minimum cycle time, the instruction execution time depends on the details of the timing of the system and must be determined on a case-by-case basis.

For a computer system where the memory access time is more than twice the central processor's minimum cycle time, the instruction execution time depends almost entirely on the memory access time. The central processor's cycle time has little effect on the overall system speed. In this case, the processor should be specified for a large and efficient instruction set to minimize the number of instruction fetch cycles, and for a large number of internal general purpose registers to eliminate the need to store intermediate results in an external "scratch-pad" memory.

The effect of the various techniques on the computer's instruction execution time may be expressed as follows: Let

T_c be the computer's instruction execution cycle time,

T_m be the effective memory read/write access time,

T_p be the central processor's microinstruction execution time,

and

M be the smallest integer that satisfies the inequality.

The internal clock of the computer is normally running continuously, so that all the computer's microinstruction cycles are synchronized to the clock ticks, which occur every 1/nth of the clock cycle (n=1, 2, 4, etc.). The effective value of the memory access time of the computer is

$$T_m = \frac{M}{n} T_p \quad n = 1, 2, 4, \text{ etc.} \tag{VI-1}$$

where

$$M \geqslant n \frac{\tau_m}{T_p} \tag{VI-2}$$

τ_m is the actual memory access time.

The computer's minimum instruction execution time (for single-memory-cycle instructions such as register-to-register arithmetic) is given for a nonpipelined computer by the formula

$$T_c = T_m + T_p. \tag{VI-3}$$

A pipelined architecture is most useful when $T_m \geqslant T_p$. In the case of a pipelined architecture,

$$T_c = T_m. \tag{VI-4}$$

If $T_m \geqslant 2\ T_p$, a split memory scheme can be used to decrease the processing time. In this case, the execution time can be cut in half. Thus, for a nonpipelined computer,

$$T_c = \frac{1}{2}\ (T_m + T_p). \tag{VI-5}$$

For a pipelined computer,

$$T_c = \frac{1}{2}\ T_m. \tag{VI-6}$$

In the case of microprogrammable processors, the time saved in performing an N-step procedure is the time it takes to fetch N-1 instructions. (One instruction must be fetched to initiate the procedure.)

Let T_s be time saved to execute the procedure, and N be the number of steps in the procedure. Then, for a nonpipelined processor,

$$T_s = (N-1)\ T_m. \tag{VI-7}$$

For a single-memory pipelined processor, the time saved is the same as above, but the time it takes to perform the procedure must be added back. Therefore, if there are no memory references in the procedure,

$$T_s = (N-1)\ (T_m - T_p). \tag{VI-8}$$

If there are memory references in the procedure, the exact amount of time saved depends on the placement of the memory references in the procedure. The maximum time is saved when the memory references occur in the beginning and are evenly dispersed throughout the procedure. The maximum time saved is, as before,

$$T_s = (N-1)\ T_m. \tag{VI-9}$$

The throughput ratio of a microprogrammed computer vs a regular computer depends on the ratio of register-to-register operations vs external memory reference operations in the procedure.

The greatest gain in processing throughput occurs when all the microprogrammed instructions are register-to-register. In this case, the regular instruction execution time would be NT_c. The speed improvement factor is

$$F_s = \frac{NT_c}{NT_c - T_s}; \tag{VI-10}$$

for a nonpipelined processor,

$$F_s = \frac{N\ (T_m + T_p)}{T_m + NT_p}. \tag{VI-11}$$

The maximum improvement factor for a nonpipelined processor due to microprogramming is

$$F_{s\ \max} = 1 + \frac{T_m}{T_p}. \tag{VI-12}$$

For a nonpipelined processor,

$$F_s = \frac{NT_m}{T_m + (N-1)\ T_p}. \tag{VI-13}$$

The maximum improvement factor for a pipelined processor is

$$F_{s\ \max} = \frac{T_m}{T_p}. \tag{VI-14}$$

If the ratio of memory access [two-cycle] instructions (as opposed to register-to-register [one cycle] instructions) to the total number of instructions is R, and R is sufficiently large that $F_s < F_s$ max, then the approximate improvement factors due to microprogramming are as follows:

For a nonpipelined processor,

$$F_s = \frac{N(1+R)\ (T_m+T_p)}{N(1+R)\ T_p + (NR+1)\ T_m}; \tag{VI-15}$$

for a pipelined processor,

$$F_s = \frac{N(1+R)}{NR+1}. \tag{VI-16}$$

The equations indicate that the minimum improvement factor in the processing speed due to microprogramming a long procedure is a ratio of two to one for a pipelined computer. For a nonpipelined computer, the minimum improvement ratio for long procedures is 4/3 if $T_p = T_m$.

The improvement factors stated apply to fairly long procedures that consist of simple register-to-register instructions and indexed memory references—machine instructions that execute with one microinstruction. When the microprogrammed procedure replaces program steps that are already microprogrammed to some extent—machine instructions that require more than one microinstruction to execute—the improvement factor is less than stated.

1. Architecture

The influence of various architectural features on computer throughput was discussed in Section III. Some features, such as pipelining and split-memory systems, influence the speed rating calculations when the speeds of the computers are compared. Other features, such as DMA access, microprogrammability, the number of general-purpose registers, processor word length, and so forth, do not affect the comparative ratings calculated above, so these must be compared independently.

2. Memory

Perhaps the most singular element affecting the performance of the computer/processor is the memory. The memory characteristics are derived directly from developed technology and are pivotal in the architecture and engineering of the computer/processor. Technology such as NMOS, bipolar, and core memories determine the size, speed, power consumption, weight, volume, and reliablity of the memory. In turn, the memory determines the computer/processor capacity, speed, power consumption, weight, volume, instruction timing, number of registers, and architecture. The software and software tools are relatively unaffected by technology, even though software does display some architectural efficiencies, as has been demonstrated in the previous sections.

In determining the memory requirements of the EW/ESM computer processor, the specifier is also determing the characteristics of the computer/processor. The specifier should go by the EW/ESM system and its signal environment to obtain his entering arguments. In the requirements (Section II), the signal environment was dictated by the altitude and sensitivity of the EW/ESM sytem. The processor throughput is then determined by these factors:

$$PRF = \frac{13264\ N_c}{T_m^{2/3}} \qquad \text{(VI-17)}$$

where

PRF = pulses per second

N_c = number of parallel CPUs (when $T_p > T_m$, $N_c = T_p/T_m$)

T_p = average microinstruction time (μs)

T_m = memory access time (μs).

This relation is based upon a minimum tracking algorithm, and as such represents an upper boundary for processors. There may be more basic algorithms that could increase throughput somewhat; however, the increase would not be a significant improvement. More sophisticated algorithms would result in reduced throughput (see Appendix A).

a. Memory size

A strong consideration for the comparison of various computers is the number of memory words that the computer can address. Some 16-bit computers are able to work only with 32,000

AD-A060 325 NAVAL RESEARCH LAB WASHINGTON D C F/G 9/2
COMPUTERS/PROCESSORS (FOR ELECTRONIC WARFARE).(U)
AUG 78 L W LEMLEY
UNCLASSIFIED NRL-8247 NL

2 OF 2
AD
A060 325



END
DATE FILMED
12-78
DDC

words of memory. While this may be adequate for small ESM systems, the designer must consider the possible need for further memory expansion, especially in the threat library, as the number and variety of potential future emitters increase.

All but the smallest ESM computers should be capable of addressing at least 64,000 words of memory. Some computers are capable of addressing up to 256,000 words or more of computer memory, making it possible to store all programming and all emitter libraries in computer core where they will be rapidly accessible.

Another factor that must be considered is the way in which memory is addressed. The more of the memory that can be addressed directly, or by indexing, the faster the program will run.

Most computers split the memories into "pages" for quick addressing. Generally, the larger the page size, the better. Page sizes smaller than 1,000 memory locations require numerous references outside the page boundaries, which slows them down.

Memory indexing can be either by page or by block, or by both. Indirect memory addressing is flexible, but slow. Because computers, whose main method of addressing memories outside the current page is by indirect addressing, will operate more slowly than computers that use indexed addressing, they should be penalized.

The size of memory is again related to the system sensitivity and the altitude of the EW/ESM or system. However in terms of the incident number of pulses per second, the memory size relation would be

$$PRF = \frac{(M - M_p + 1)\ C_{ws}}{P_{ws}} \left[\frac{N_c}{N_i \sum_{i=1}^{n} w_i t_i} \right] \qquad \text{(VI-18)}$$

where

PRF = maximum pulse density

M = memory size (bits, bytes, words)

M_p = program size (bits, bytes, words)

C_{ws} = computer's word size (bits, bytes, words)

P_{ws} = data word size (bits, bytes, words)

N_i = number of microinstructions

t_i = execution time of ith microinstruction

N_c = number of processor channels (multipoint memory)

The program size typically is as discussed in Section IV. Current emitter environments are related to system sensitivity and altitude as noted in Section II. Emitter characterization was discussed in Section III and subsequently in Section VI.A.5 under word size.

b. Memory speed

It is apparent that for processors operated by a constant-frequency clock, the minimum clock period of the pipelined processor is determined by either the memory access time or the central processing unit's microinstruction cycle time, whichever is greater. Therefore, if the memory access time is shorter than the CPU's basic cycle time (less the address set-up time), the operating throughput of a computer is determined almost entirely by the basic cycle time of the central processor. On the other hand, if the memory access time is a bit larger than the CPU's basic cycle time, the operating throughput is determined by the memory access time (plus the address set-up time). [12]

In order to operate with even slower memories, the system designer has two options—either take more than one clock cycle to access the memory, or stop the clock entirely and have the memory acknowledge when it is ready. In either case, the clock is run at the minimum cycle time of the microprocessor, but the total instruction execute time is determined largely by the memory access time, since only a very few instructions require a large number of execute cycles without accessing the memory for more data or further instructions.

An example of a regular processing sequence in which the processor takes two clock cycles to access memory is shown in Fig. III-3c. In practice, most processor clocks do not run asynchronously, so the processor does not start as soon as the memory acknowledges that it is ready, but waits till the next clock tick (which may be either a quarter, half, or full clock cycle, depending on the CPU and the clocking scheme used). If the memory is so slow that the memory access time is more than twice the microinstruction cycle time, some of the lost processing time can be recovered by using a "split memory." Memory access time is technologically related; however, there is a high correlation with memory size due to propagation path. Figure VI-1 presents a general relationship between memory access time and memory capacity. This relationship changes as technology improves. Figure VI-1 represents 1977 technology.

c. Split-memory system

In the split-memory scheme, the computer memory is split into odd and even sections, each section being independent of the other. Each memory section is addressed by its own independent memory address register at the central processor unit, and each has an independent (or multiplexed) address bus.

The pipelined instruction timing for a split-memory computer is shown in Fig. III-3d. Each memory bank is fully pipelined. The two memory banks are addressed out of phase so that the instruction execute period that would normally be wasted in a regular pipelined system is used to execute the instruction from the other memory bank.

This technique nearly doubles the instruction throughput of a fast processor coupled to a very slow memory. It is used extensively in modern military computers, which must marry ultrafast microprocessor-based central processors to nonvolatile but slow magnetic-core

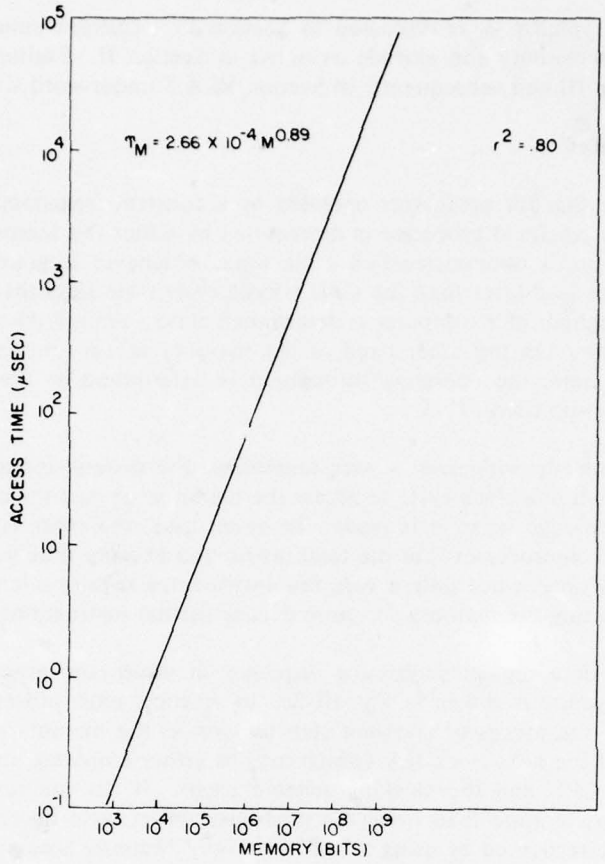


Fig. VI-1—Access time vs memory capacity

memories in order to meet military specifications for nonvolatile memories. The execution speed of these computers depends almost entirely on memory access speed.

d. Direct memory access

If the computer is operating in a regular nonpipelined sequence, with the instruction fetch and the instruction execute cycles being the same length, then data can be entered into the computer memory under DMA control without affecting the operation of the computer, as shown in Fig. III-4b.

If much data must be transferred very rapidly, the main computer can be briefly halted, and data transfer can take place continuously as shown in Fig. III-4c. The advantage of DMA transfer over computer-controlled I/O transfer is that during computer-controlled transfer, the computer executes its normal instruction fetch and instruction execute cycles to perform the transfer, whereas under DMA control the data transfer occurs continuously.

In a pipelined computer without a split memory, DMA transfer cannot occur without interfering with the operation of the computer. The computer must be slowed down while DMA data transfer is taking place as illustrated in Figs. III-3b and III-4b.

3. Timing

a. Regular

A typical microinstruction execute cycle is shown in Fig. VI-2. On the positive edge of the clock, the central processor receives the microinstruction, decodes the control word, selects the operands, performs the given operation, and selects the destination of the resultant. On the negative edge of the clock, the resultant is stored in the proper locations and the CPU waits for the next control word, which may specify another operation or may begin another instruction fetch cycle. [6]

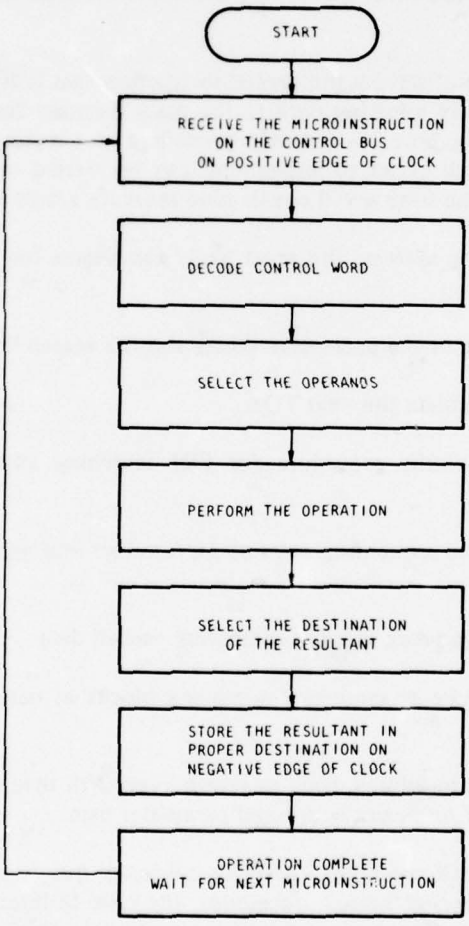


Fig. VI-2—Microinstruction execution cycle

b. Pipelining

Under this scheme, it is apparent that the CPU is idle during the instruction fetch cycle, whereas the memory is idle during the instruction execute cycle. A technique to interleave two consecutive instructions, called pipelining, is illustrated in Fig. III-3b. [6, 10—12]

The algorithms employed in ESM processing usually contain a large number of compare and branch instructions, so the full potential increase in execution speed of the pipelined architecture cannot be realized in ESM processors.

4. Microprogrammability

A microprogrammable computer allows users to insert their own control and sequencing microinstructions into the microprogram memory of the CPU. This gives the computer a set of unique user-defined instructions for performing special job-related repetitive or time-consuming procedures. [12, 15]

The major advantage of having the special instruction sets is that these procedures can be executed without continually referring back to the main memory for instructions, thereby saving considerable processing time. A ten-step procedure, for example, which would normally require ten instruction fetch cycles to implement, can be carried out after a single instruction fetch cycle. In this case, the time saved equals nine memory access times (see Appendix A).

In an ESM processing system, the most likely candidates for microprogramming are the following procedures:

- The adaptive tracking of the parameter values and the search limits update procedures
- The procedure to calculate the next TOA
- A compare-between-limits procedure for PRI searching and for emitter identification searching
- The PRI procedure for comparing pairs of TOA values and generating the next TOA window
- A masked comparison procedure for comparing packed data
- Multiple load and store procedures for moving blocks of data between internal registers and memory
- Special table-search procedures, such as search every Nth byte or every Nth word, to facilitate the searching of numerically ordered parameter lists.

The increase in overall computer operating efficiency due to microprogramming depends on the length of the microprogrammed procedures, the ratio of internal-to-memory access steps in the procedure, the ratio of the microinstruction cycle time to memory access time, and the fraction of time the procedure is used during normal processing. The gain in operating speed

when using the microprogrammed procedure over the regular procedure is typically 3 to 1. A comparison of processor throughput for a microprogrammed computer vs a regular computer in performing the adaptive tracking task is analyzed in Appendix A. The processing throughput improvement varied from 2 to 1 through 4 to 1 for the various processors investigated.

Formulae for the increase of processing throughput due to microprogramming are presented in Appendix A.

Optimization tables for microinstructions to increase EW/ESM throughput are presented in Section IV.A,1.b. Tables IV-6 and Fig. IV-2. Next to memory access time, the optimization of these microinstructions would have the greatest effect upon EW/ESM throughput in the main computer. For preprocessors the reader is referred to Appendix A.

5. Word Size

In order for the computer to operate at a high speed, it is important that the computer word length not be too small. The time required to fetch a word from memory is essentially independent of the word length. Fetching a two-word instruction, however, takes about twice as long as fetching a one-word instruction. Therefore, from the point of view of the instruction set, a computer word must be long enough to allow all but a few of the instructions to be specified in single instruction words. On the other hand, instruction word bits not used are wasted, so that overly long instruction words simply increase the cost without increasing the performance.

The cost-vs-performance tradeoff has been studied by various minicomputer manufacturers, with the result that the majority of the manufacturers have adopted the 16-bit computer word size as a standard. These 16-bit computers can perform all the fast register-to-register operations with one-word instructions, although most computer operations that address the main memory directly require two-word (32-bit) instructions. Special addressing techniques, such as indexed addressing and memory paging, permit some main memory operations with one-word instructions. It may be considered that a 16-bit processor word is a reasonable minimum word size for the ESM processor instruction words, while a 32-bit word size is preferred if the processing throughput rate must be a minimum.

These same arguments hold for data words, except that for some mathematical operations, such as shifts, data stored in two short memory words takes *more than twice as long* to process as it would take to process the same data if they were stored in one long memory word. This is one reason why large number-crunching computers tend to have 48- to 64-bit memory words. On the other hand, data words that are longer than required waste memory.

It is possible to increase the data storage efficiency by packing more than one data word into each of the extra-long memory words. However, for very short data words, a significant portion of the computing time gets spent in packing and unpacking the data bits, and the advantages of using these extralong memory words are lost.

The optimum data word size for the ESM System can be determined from an analysis of the amount of information needed to uniquely locate and identify each emitter of interest in the environment. The actual number of required data bits depends on the proposed data

Table VI-1 — Possible Parameter Word Size Requirements

Parameter	Range	Increments	Word Size (bits)
Angle of arrival	0—360°	0.35°	10
Time of arrival	0—200μs	0.2 μs	20
Signal amplitude	−120 to +8 dBm	1.0 dBm	7
Pulse width	0.25 μs	—	—
	0—12.5 μs	0.1 μs	8
Frequency	0—5 GHz	5 MHz	
	5—10 GHz	10 MHz	11
	10—15 GHz	20 MHz	
or			
Frequency	2—4 GHz	1 MHz	
Total			56

analysis algorithm, which, in turn, depends on the specific application envisioned for the ESM System.

An estimate of the number of bits that should be sufficient to represent each of the various signal parameters is presented in Table VI-1.

In selecting the processor data word size, an attempt should be made to avoid splitting up any of the parameter data words into two processor words. Each portion of the split word must be processed separately, and then the separately processed portions must be logically connected together with further processing; thus a data word, split in two, takes more than twice as long to process as a single data word. For example, in Table VI-1 the minimum processor data word size is 20 bits to accommodate the 20-bit TOA word. All data for one emitter pulse could be stored in three 20-bit words, two 28-bit words, or one 56-bit word.

In most general-purpose (GP) computers, the processor instruction word size and the data word size are the same. This allows each GP memory location to be used to store either data or instructions, whichever a given program may require. Since a large variety of programs may be run on a GP computer, this arrangement results in the most efficient utilization of main memory.

In a special purpose ESM processor dedicated to performing only one given task, the instruction memory and data memory can be kept separated, with each type of memory having a different word length suited to the given application.

During the evaluation of five military family architecture computers by MCF (Section IV), the high EW/ESM correlation Benchmark programs were examined for word-size efficiency. In the memory activity index M, there was an 18% loss of efficiency in going from a 16-bit word to a 32-bit word. However, in the register activity index R, these same Benchmarks displayed a 12% increase in activity in going from a 16-bit word to a 32-bit word. Apparently there is some gain to be realized in memory packing efficiency with a 16-bit word, however, processing with 32-bit words brings a 12% improvement in efficiency.

6. General-Purpose Registers

General-purpose registers [15,16] are useful for storing intermediate results of mathematical operations, as index registers for modifying addresses for repetitive (looping) calculations, as pointer registers (page pointer) for addressing different blocks of data, for block transfer of data from location to location, or as multiple accumulators for arithmetic or logical functions. The number of GP registers in a processor is usually chosen to be a power of 2 from binary addressing considerations.

Register-to-register operations are normally performed in a single microinstruction cycle, so a large number of GP registers can provide for extremely rapid data processing by decreasing the number of external memory references required. Register-to-register instructions are usually constructed in single-word format, which decreases program storage space and also increases the processing speed.

General-purpose registers are sometimes arranged in two banks, with each bank independently available for data storage. Programs called repeatedly into operation and applications requiring rapid switching from one task to another can each be assigned a register set for its own use. This increases processing speed by eliminating the need to change the contents of the general registers each time a task is changed.

The relationship of registers to throughput is discussed in Section VI.A and Appendix A. The number of registers that would optimize throughput is program dependent.

7. Special Arithmetic

Programming versatility can be enhanced and computation throughput increased in some processors by using separate IC hardware to perform special functions that would take very long to perform with software [14—16]. The hardware often includes the following functions:

- Floating-point arithmetic hardware, which performs precision floating-point instructions much faster than ordinary floating-point software subroutines
- Double-precision multiply and divide for direct processing of double-length operands
- Trigonometric function package which includes sin, cos, arctan, and other trigonometric functions.

In ordinary operation, most ESM processors do not make extensive use of floating-point or double-precision arithmetic, so these functions need not be included in an ESM processor as special hardware. The trigonometric function package could be useful for navigation and for emitter location algorithms if these calculations are performed fairly often. If the calculations are done infrequently, they may as well be implemented in software.

Some newer ESM Systems are turning to sophisticated mathematical techniques, such as adaptive Kalman filtering, to perform the adaptive tracking algorithms and fast Fourier transforms to do the PRI processing. These techniques normally take too much processing to allow a high processor throughput if they are implemented in software. They may prove to be practical, however, if performed by adding special function hardware to the computer.

8. Input/Output (I/O)

A large number of modern computers are actually multiprocessors, employing a CPU to perform the main work of the computer and using special purpose microprocess-type computers to handle peripheral chores. The most common use for an additional processor is in the implementation of an I/O controller.

A separate I/O controller executes I/O instructions independently of the central processor. Data are transferred between the I/O controller and main memory by stealing memory access cycles from the central processor. Otherwise, the I/O controller operates independently to interface a large number of peripherals using a variety of standard parallel or serial interface types.

9. Multiprocessing

The system designer must decide how to allocate the computer resources that will do the processing. The obvious first approach would be to use a big GP computer to do all the processing. In this case, all processing would be done in one CPU, and all data would be stored in one central memory, as shown in Fig. III-5.

As pointed out in Appendix A, the entire processing capability of an AN/UKY-20 was not enough to track more than ten emitters on a pulse-to-pulse basis, so it becomes apparent that at least some processing will have to be done by hardware, or special-purpose software processors.

The approach taken by most modern ESM Systems is to divide the processing task among as many processors as possible. Since each processor works in a small segment of the problem and all processors operate simultaneously, the overall system throughput can be increased considerably by using multiprocessing. Task size and processor speed are selected so that each processor works at about the same rate, enabling the data to flow smoothly through the system, with no backup.

The system suggested in Section III (Fig. III-5) is such a system. The approach shown, where no more than two processors access each data memory, can be designed with minimal interference between processors by interleaving the data transfer timing of the two processors. Since each processor accesses two data memories, as well as its own individual instruction set memory, the processors can use pipelined architecture most of the time. Unfortunately, this technique wastes memory space and loses some of the processing time, since duplicate data must exist in several memory blocks and a part of the processing time must be used to merely move data from one memory block to another.

10. Physical Characteristics

Other factors that must be considered are the physical size, weight, and power drain of the computer. If the computer is to be part of an airframe, the size, weight, and power drain become important considerations in its selection. If the computer has to be used on the ground, size is usually secondary.

A large size computer can be an advantage if it leaves room for plug-in options that the designer wishes to incorporate. Computer and memory options that are "add-ons" require a separate chassis, separate power supplies, and additional cabling, all of which increase the cost and decrease the reliability.

The minimum boundary physical characteristics of a processor are also closely correlated to memory size, whether they be military qualified or commercial. Of course, the minimum boundary commercial characteristics outperform military values due to the ruggedization requirements. These values may change with technological improvements that are addressed in future processor projects; however, at present (ca. 1977) military processor power requirement performance is

$$Pwr = 36.3 + 4.8M, \tag{VI-19}$$

where

Pwr = power in watts

M = static memory capacity (16-bit kilowords)

and $\pm 1\sigma$ is

$$5.4 + 4.5 < Pwr < 67.1 + 5.0M. \tag{VI-20}$$

The relationship between power and memory capacity is illustrated in Fig. VI-3 with a benchmark representing the size of one AN/ATR box.

Processor weight, in the same manner, is related to memory size, as

$$W_p = 3.7 + 0.69M, \tag{VI-21}$$

where

W_p = weight in pounds

and $\pm 1\sigma$ is

$$-16.5 + 0.59\ M < W_p < 9.2 + 0.79M. \tag{VI-22}$$

The appearance of a negative value (weight) for low memory size is obviously invalid and due to the statistical method of analytically representing processor weight related to memory size. However, when the total expression is examined, the upper 1-σ limit allows a low memory—weight relationship.

This relationship is illustrated in Fig. VI-4 with the volume of a full-size ATR box as a benchmark.

Finally, the volume relationship also correlates to memory size, as defined above, as follows:

$$V_p = 369 + 7.67\ M, \tag{VI-23}$$

where

V_p = volume in cubic inches

and $\pm 1\sigma$ is

$$187 + 6.29\ M < V_p < 551 + 9.05\ M. \tag{VI-24}$$

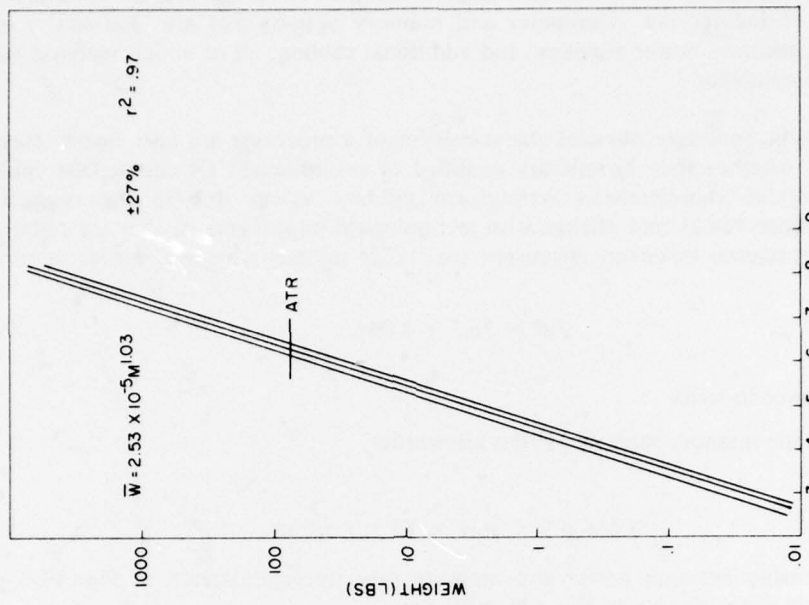


Fig. VI-4—Weight vs memory capacity

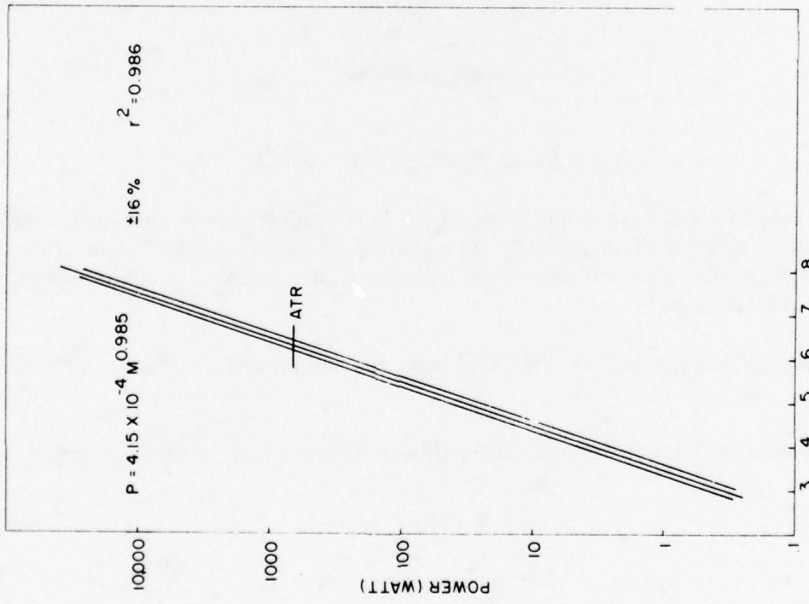


Fig. VI-3—Prime power vs memory capacity

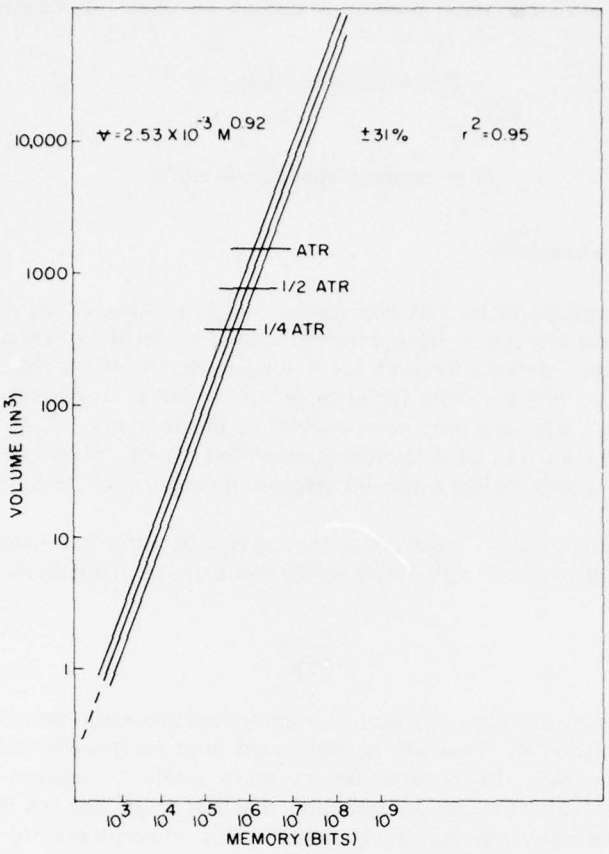


Fig. VI-5—Volume vs memory capacity

As previously, these relationships are illustrated in Fig. VI-5 for volume vs memory capacity. The volume of various standard-sized ATR boxes are marked as benchmarks for easy reader reference.

The above relationships all apply to military standard experience (ca. 1977) in which generally airborne processors were analyzed. Airborne processors were chosen since weight, size, power are more critical to this platform (satellites being grouped with airborne). The rationale is that performance designed to airborne requirements more nearly represents what the state of the art can accomplish in these characteristics.

Evaluating physical characteristics of more advanced commercial (or R&D) products becomes a bit more speculative as may be expected, since no attempt has been made to optimize their packaging. However, some insight may be achieved by comparing the pertinent factors that contribute to these physical characteristics at the commercial and R&D level. For instance,

present commercially available static memories display an order of magnitude improvement over standard military;

$$Pwr = 0.334 + 0.0363\ M, \tag{VI-25}$$

where

M = memory capacity (kilobits).

11. Reliability and Maintainability

Reliability is important in military applications; all else being equal, computers that have been in production long enough to have demonstrated their reliability should be given preference over new designs. Memory failures are a serious source of computer malfunctions, so memories that use extra bits per word for error detection and error correction should be more reliable than computers that use only working bits in the memory. A 16-bit-word computer requires only one extra bit for the detection of single-bit errors. Five extra bits added to the 16-bit computer word would correct single-bit errors and detect multiple-bit errors.

To increase maintainability, most computers have diagnostic fault-isolation software, and most of them are constructed of easily replaceable modules, so maintainability should not be a serious problem.

B. Software

The major factor in the life cycle cost of a computer/processor-based (EW/ESM) system is the available software [19]. This will be illustrated both analytically and empirically in the subsequent section on costs. In the past, far too many small- to medium-scale military computer systems were developed using the computer that was to go into the final operational system as its own software development environment. The consequences of this approach were often disastrous, since these development environments were virtually devoid of the very broad spectrum of powerful support software tools that now exist on larger computers.

The software aspects of (EW/ESM) computer/processors have been covered quite throughly in Section V. It remains for this section on computer/processor specifications to emphasize the requirement for well-developed software tools. Consequently, in Section VIII the cost of these tools will be discussed. It will be up to the specifier to determine whether he is prepared to support those costs or would be prepared to "adapt" his processor need to less costly, "established" architecture. For the most part, this section is oriented to the requirements of EW/ESM; however, as has been implied in the previous and ongoing discussion, the methodology and conclusions are generally applicable to both military and commercial requirements.

1. Software Tools

In Section V, software tools were analyzed for their relative contributions to EW/ESM requirements through correlation with machine EW/ESM performance. The following discussion breaks these tools down into three statistical catagories: (a) high EW/ESM contribution, (b) medium EW/ESM contribution, and (c) low EW/ESM contribution. Within these general

categories, the software tools as listed below, in order of decreasing EW/ESM contribution, although the differences may be small.

High Contribution (greater than 14%)

1. Language-Independent Monitor
2. Noninteractive Debugging Aids
3. Compilers and Cross-compilers

Medium Contribution (±14%)

1. Test Case Intrumenters and Analyzers
2. Test Case Auditor
3. Test Case Generator
4. Text Processing System
5. General-Purpose System Simulator
6. Instruction Simulator
7. Assembler
8. Macro Assembler
9. Basic Linker
10. Interactive Source Editor
11. Batch Source Editor
12. Intregrated Library
13. Language-Dependent Monitor
14. Reformatter
15. Real Time and Time-Sharing Operating Systems
16. Data Base Management System
17. Interactive Debugging Aids
18. Computer System Simulator
19. Real Time and Time-Sharing and Virtual Machine Monitor Operating System
20. Time-Sharing and Multiprocessing and Virtual Machine Monitor Operating System

Low Contribution (less than 14%)

1. Extended Overlay Linker
2. Data Base Design Aid
3. Simple Overlay Linker

The CFA listed these tools in "layers" according to their general military utility. These layers and a short description of the tools are described below. [20]

Layer 3: Functional Support Tools

Layer 3 contains those tools that provide direct support to the software development activities. The applications software developer has the greatest interaction with these tools. Layer 3 tools will be related to the specific development activities they support.

Layer 3: Tool Types That Support Activity 1 (Analyze Requirements)

The types of tools directly applicable to requirement analysis are GP system simulations and system description languages and analysis.

• General Purpose System Simulators—These allow a user to construct a computer model of a real or proposed system and to perform simulations to determine the behavior of the model under various operational conditions.

• System Description Languages & Analyzers—These assist system analysts in describing the functional characteristics of a system and in validating the consistency and completeness of a functional decomposition.

Layer 3: Tool Types That Support Activity 2 (Design Software)

The types of tools directly applicable to software design are listed below.

• Computer System Simulators—These are similar in nature to the general purpose simulator except that their basic building blocks represent real computer system components whose modeled behavior approximates the throughputs, capacities, and access times achievable on the modeled equipments.

• Data Base Design Aids—These aids assist data base designers in grouping data elements into logical record classes and in determining the relationships among logical record classes implicit in either the nature of the data or the usage of the data.

• Data Dictionary Systems—These systems assist data base designers in managing the data definition activities.

Layer 3: Tool Types That Support Activity 3 (Build System Tests)

The types of tools required to support system test construction are listed below.

• Test Data Generators—Create data files for testing and validating programs.

• Test Data Auditors—Compare data files against specification and produce reports of discrepancies and/or compliance.

• Test Case Design Advisors—Analyze programs written in a high-level language and present the results of that analysis in a form suitable to assist test-case designers in the selection of test data.

• Test Instruments and Analyzers—Instrument modules under test so as to collect data characterizing the behavior of the module.

Layer 3: Tool Types That Support Activity 4 (Build and Unit-Test Software)

The types of tools required to support the program development and unit-test activity are listed below.

• Assemblers—These allow programs to be coded in a symbolic language in which statements generally correspond to a single machine instruction. Specific tools include basic assemblers and macro assemblers.

• Compilers—Translate programs written in a high-level language into either relocatable object code acceptable to a linker or an assembly language acceptable to an assembler.

• Linkers—Combine the text produced by separate interrogations of compilers and assemblers ("object modules") resulting in executable code strings ("load modules" or "core images") that can be loaded into the computer's main storage and executed without further preprocessing. Specific tools are basic linkers, simple overlay linkers, and extended overlay linkers.

• Debugging Aids—Assist the programmer in locating the sources of program errors that have been discovered during unit testing, usually by giving him some control over the execution of the module under test that is external to the normal program code. Specific tools are interactive symbolic debuggers, noninteractive symbolic debuggers, interactive absolute debuggers, and noninteractive absolute debuggers.

• Module Libraries and Change Control Systems—Provide computer-controlled maintenance of groups of related source modules (programs), object modules (the output of assemblers and compilers), and load modules (the output of linkers). Specific tools are basic libraries, integrated libraries, and automatic software production and test systems.

• Performance Monitors—Assist the programmer in quantifying the resource consumption characteristics of a program and in isolating performance-critical areas. Specific tools are Language-Dependent Monitors and Language-Independent Monitors.

• Standards Enforcers—Allow source programs to be examined automatically and checked for conformance to installation-defined standards of format, content, and usage.

• Preprocessors and Reformatters—Assist programmers in producing well-structured and readable programs by allowing the programmer to introduce to structured programming elements into source programs for languages that do not have them, and by automatically controlling indention, the placement of comments, etc., to produce readable listings.

Layer 3: Tool Types That Support Activity 5 (Integrate and System Test) and Activity 6 (Maintain System)

No unique layer 3 tools exist to support these activities. The tools that were listed for activities 1 through 4 are generally appliable to activities 5 and 6 at layer 3. Most of the tools used in practice that are specifically oriented to activity 5 are special-purpose, e.g., test environment tools (emulators, hot benches, system integration lab support, virtual machines), test drivers and special performance monitors.

Layer 2: General Support Services

The primary function of layer 2 tools is to provide a framework of common services that will allow the output of third-layer functions to be stored, retrieved, and intercommunicated. Second-layer functions should be usable for common purposes across different third-layer functions, and should serve to hide (where possible) differences between first-layer and third-layer functions. Layer 2 tool types provide general support to all of the software development activities. These tool types are summarized as follows:

• Data Base Management Systems—Allow the use of a computer system to define the contents of and the logical relationships between collections of data items that represent some useful abstraction of a real-world phenomenon (tactical command and control system, and modules and documentation of a system of computer programs) without being concerned with the physical mechanics of storing, locating, and retrieving items or groups of items.

• ERT/CPM System—Assist managers in planning and controlling project activities.

• Project Estimation Systems—Assist in the development of work breakdown structures and related performance standards for use in estimating project resource requirements.

• Documentation Aids—Assist in the preparation and maintenance of documentation about the modules of a system. Specific tools are text processing systems, flowchart construction languages and automatic flowcharters.

• Data Manipulation Utilities—Allow the system user to alter the format and content of data files independently of the logical significance of the data fields involved. Specific tools are sort/merge programs and editors (interactive source language editors, interactive object module editors, batch source language editors, and batch object module editors).

• Information Retrieval Systems—General purpose application programs operating either on-line (interactively) or in-batch that interpret user requests to locate and display information that is stored either within a structured data base or within separate files: Specific tools and query language systems and report writers.

Layer 1: Operating System Services

Layer 1 implements the operating system services that present a "virtual machine" interface to the services/tools at layers 2 and 3 and manage the real system hardware. The layer 1 tool types are generally applicable across all of the software-develpment activities. Layer 1 tool types/capabilities are listed below:

• Basic Operating Systems (BOS)—Run single-user processes from initiation to termination. May or may not overlap I/O with execution. Provide basic I/O support that allows user to refer to files symbolically and to read and write them without knowing the hardware details of the I/O interface. Provide basic batch supervisor services that control normal and abnormal job termination, job-to-job transition, and operater communication. Provide a minimum base for program development by supporting at least one language translator and/or linker/loader.

• Multiprogramming Operating Systems (MOS)—Provide all of the services of the basic operating system. Supports the concurrent execution of two or more user jobs by allowing the execution of any job to be suspended while another is executed, without any special programming considerations in the user job. Prevents concurrently executed user jobs from accidentally or intentionally destroying each other or the supervisor.

• Multiprocessor Operating Systems (MPOS)—Allow the computing load to be spread across more than one processor; based on automatic (programmed) load-leveling algorithms or operator control, but does not require special case programming in the user job. Multiprocessor Operating Systems include the shared storage, loosely coupled, and networked types.

• Virtual Machine Monitor (VMM)—The operating system presents an interface to the user program that makes it appear that the program is executing on a real computing system.

• Time-Sharing Operating System (TSOS)—This is a variant of the multiprogramming operating system in which system resources are allocated to user jobs in such a way that all jobs appear to progress at the same rate. In addition, users are allowed to "interact" with and receive outputs from their jobs via terminals. Such systems are optimized for response rather than throughput.

• Real-Time Operating Systems (RTOS)—Allow user jobs to be executed within specified short time limits.

A number of approaches can be made to the specifying software tools:

a. *High-Cost Approach.* The specifier may describe each tool as required for the processor that is being processed and require the vendor to develop this tool(s) for the architecture processor being procured.

b. *Medium-Cost Approach.* The specifier requires the vendor to conform the processor architecture to operate with established operational software tools.

c. *Low-Cost Approach.* The specifier stipulates that the vendor develop his computer/processor to comform to an established software base, preferably from an established high-inventory computer architecture.

2. Software Language

One conclusion that can be established from Table V-2 of the preceding section on software is that the military language, CMS-2, appears to enjoy a 78% efficiency advantage over the more common FORTRAN. However, in general military applications, FORTRAN does appear to be more efficient than CMS-2 by 18%. Another factor that will be established in Section VIII regarding cost is that the FORTRAN base will be on the average 34% less costly to program than CMS-2 because of the higher base of programmers familiar with the FORTRAN lauguage.

C. Microinstructions

Section IV, under *Evaluations,* established that EW/ESM programs used 10 of 103 microinstructions nearly 80% of the time. It would follow that the specifier should require that any computer/processer procured for an EW/ESM system be optimized for at least these ten instructions. Optimization in this case would be in the form of execution times. These instructions, together with their relative weighted importance, appear in Table VI-2.

Table VI-2 — Weighted Priorities for EW/ESM Microinstruction

Instruction Category	Frequency	Cumulative Percent	Weight
1. Load Internal Register	3289	21.4	.271
2. Store Internal Register	2372	36.8	.195
3. Quick Branch	1598	47.2	.132
4. Branch	1043	54.0	.086
5. Branch to Subroutine	0834	59.5	.069
6. Register Add/Subtract	0812	64.7	.067
7. Branch on Condition	0622	68.8	.051
8. I/O Control	0587	72.6	.048
9. Register Logical Operation	0541	76.1	.045
10. Modify Internal Register	0454	79.1	.036

VII. SELECTION

As part of the specification process, the computer/processor procurement must be capable of selecting the best technical response to his request for proposal. The vendor, in turn, should expect and derive a great deal of information from the basis of selection. The selection process is a weighting process by which every element of importance in the selection of a computer/processor is weighted for its contribution to the expectations of a system; in this case, an EW/ESM system.

As implied by this ongoing text, the criteria of selection are complicated and extensive. The most important criterion is memory access time; however, most modern developed computer/processors should be very close in technology, so that this single characteristic would not provide sufficient basis for choice. A second criterion of performance is the microinstruction execution time as discussed in Section IV. Here, individual hardware architectural features will begin to exhibit differences that will permit choices of performance, especially EW/ESM performance. Software architecture and software tools become a third important criterion of choice. Without the software tools the computer/processor becomes an extremely frustrating, time-consuming and costly choice, even if it does exhibit some performance superiority. Hardware costs are often diminished by the high-cost, programmer-related expenses of software. Life-cycle costs are dominated by the software cost over the initial cost of hardware acquisition, as will be shown in Section VIII.

This is not to say that cost is an overriding criterion of selection. There are a number of reasons why performance must override cost; however, within the boundary conditions of performance, these criteria may be used for selection. The purpose of this document and section is not to make a selection but to provide the basis by which selection can be made, given the normal objectives of (EW/ESM) system processor procurement.

A. Memory Access Time

As has been pointed out in several areas of this document, memory access time is the sole criterion of real-time throughput performance. Section IV demonstrated the performance

differences that can be realized for computer/processors of the same architecture when memory access time is changed through technology. Table VII-1 compares two computers of the same architecture with different memory technologies (core, semiconductor, bipolar). The (EW weighted average) instruction execution time dramatically demonstrates the performance differences for the change of memory even though the same architecture is retained.

Table VII-1 — Computer Architecture Performance vs Memory Access Time

Computer	Memory Access (ns)	Instruction Execution (μs)
ATAC-S	650	2.71
ATAC-F	200	1.15
PDP-11/45 C	452	2.245
PDP-11/45 B	300	0.986

Generally, the average instruction execution time is related to memory access time by

$$T_p = 1.824\ T_m^{2/3} \qquad \text{(VII-1)}$$

where

T_p = average instruction execution time (μs)
T_m = memory access time (μs).

However, differences do exist, due to the particular architecture, so that the next level of comparison for performance selection lies in a (EW/ESM) weighted comparison of the most frequently used microinstruction execution times.

B. Instruction Execution Time

Selection between computer architectures on a hardware basis is carried out by a comparison of the architecture's microinstruction execution times. These factors depend upon the particular usage the program makes of these instructions. As was pointed out earlier in Section IV on *Evaluation*, EW/ESM programs are almost dominated (80%) by ten microinstructions that represent the frequency-of-use times execution-time product of an EW/ESM program (see Table VII-2). It is not only sufficient to note the frequencies of microinstruction usage; but, since each instruction is executed with different times, the frequency—time product is more indicative of the architecture's performance than frequency alone.

The use of weighted microinstruction selection permits a performance evaluation based on not only the previously discussed memory characteristics, but also the architectural structure of the CPU, ALU, bus, registers, and all elements that would contribute to the performance data for the computer/processor. The computer/processor vendor, then, should provide the performance data for his candidate processor as against the microinstruction weighted list of Table VII-3 for EW/ESM application. The absence of any microinstruction would have a penalizing effect upon the particular machines performance in an EW/ESM system. (Table VIII-4 presents the variations in microinstruction execution times realized by various computer/processor configurations.)

Table VII-2 — Ten Most Frequently Used
Frequency—Time Product Microinstructions
for EW/ESM Programs

Instruction Type	Frequency	Cumlative Percent	Frequency — Time Product
Load Internal Register	3289	21.4	25517
Store Internal Register	2095	35.0	17688
Quick Branch	1598	45.4	5321
Branch	1043	52.2	6519
Branch to Subroutine	834	57.7	5738
Register Add/Subtract	812	62.9	6431
I/O Control	587	66.8	7396
Branch on Condition	586	70.6	4213
Halfword Logical Operation	541	74.1	3522
Modify Internal Register	454	77.1	3673

Table VII-3 — Weights of Ten Most-Used
EW/ESM Microinstructions

Instruction Category	Weight
Load Internal Register	0.271
Store Internal Register	0.195
Quick Branch	0.132
Branch	0.086
Branch to Subroutine	0.069
Register Add/Subtract	0.067
Branch on Condition	0.051
I/O Control	0.048
Register Logical Operation	0.045
Modify Internal Register	0.036

C. Benchmark Tests

Several levels of Benchmark tests are available that will produce various degrees of accurate comparisons. Appendix A provides the details of a simple EW/ESM tracking algorithm that can benchmark test the most fundamental processor either programmatically or analytically, as demonstrated in Appendix A. More software-oriented, the CFA/MCF programs generated a series of Benchmark programs to test the elements of (software) computer architecture that most represent the needs of general military computer/processing. These benchmark programs were analyzed through correlation for their ability to characterize EW/ESM computer/processor performance. One CFA/MCF Benchmark program, 1. Message Buffering and Transmission, displayed very high S, M, and R correlation with the EW/ESM performance of the computers on which they were tested. Other Benchmark programs displayed individually higher S, M, and R correlations; and in some cases, the Benchmark programs displayed no correlation at all.

One problem with the CFA/MCF Benchmark programs is that they are subject to individual programmer interpretation and implementation such that programmer variances are

involved. This results in a less rigid approach to computer/processor selection than the above microinstruction approach or the flywheel tracking algorithm of Appendix A. On the other hand, the implementation of a microinstruction analysis or program is much more difficult than the higher level programming demanded of the CFA/MCF Benchmark tests. The results of the CFA/MCF test must almost be treated statistically, as they were in that program. On the other hand, the CFA/CMF Benchmark programs do test the higher levels of (software) architecture and would permit an evaluation of software tools and programmer efficiency if demanded in the specification and selection process.

Table VII-4 — Mean and Standard Deviation for Microinstruction Execution Times

Microinstruction	Time, t (μs)	$\pm\sigma$	$\pm$%
LAH	2.289	1.321	57.7
MIC	1.398	0.900	64.4
SAH	2.438	1.401	57.5
B	2.129	1.044	49.0
BSI	3.371	1.708	50.7
LYR	1.833	0.858	46.8
SA	3.128	1.485	47.5
LA	3.086	1.225	39.7
MXR	1.622	1.141	70.4
SBZ	2.914	1.442	49.5
DIOC	4.181	2.886	69.0
AUA	1.662	0.891	53.6
SR	4.114	2.453	59.6
AH	1.859	1.456	78.3
BZ	2.245	1.226	54.6
SHZ	1.951	1.554	79.6
SHN	2.023	1.736	85.8
SH	1.859	1.456	78.3
MSH	4.487	1.607	35.8
SXR	2.886	1.605	55.6
SL	4.400	4.163	94.6
$\bar{t}$	2.465	1.322	53.6

Table VII-5 presents the Benchmark programs together with their EW/ESM relevancy as determined through correlation with the EW/ESM performance of known computer/processors. The correlation coefficient was determined from the relation:

$$r^2 = \frac{\left[\Sigma(\ell n\ x_i)\ (\ell n\ y_i) - \frac{(\Sigma \ell n\ x_i)\ (\Sigma \ell n\ y_i)}{n}\right]^2}{\left[\Sigma(\ell n\ x_i)^2 - \frac{(\Sigma \ell n\ x_i)^2}{n}\right]\left[\Sigma(\ell n\ y_i)^2 - \frac{(\Sigma \ell n\ y_i)^2}{n}\right]} \qquad \text{(VII-1)}$$

where

x_i = the EW/ESM index of performance of the ith machine
y_i = The Benchmark program value on the ith machine.

Table VII-5 gives the EW/ESM correlation relevancy values for program size, S; memory activity, M; and processor activity, R. The values are most relevant as they approach 1.000, which is perfect correlation. Values approaching 0.000 have practically no relevancy at all and would not normally be used to select a computer processor for EW/ESM. In some cases correlations are negative; that is, while relevancy exists, there is an inverse relationship between the EW/ESM index and the Benchmark program value.

The S, M, and R values that the MCF program derived from Benchmark tests were correlated in a similar manner with the EW/ESM performance of the architectures tested. This yields a fairly reliable relationship between the S, M, R values and the EW/ESM index such that performance can be projected based upon S, M, R rather than the EW/ESM index when one or the other is not available:

$$T_p = 2.22 - 1.094M + 2.298R - 1.624S, \qquad \text{(VII-2)}$$

where

T_p = EW index (or average EW weighted processor time)
M = memory activity index
R = processor activity index
S = program size index.

D. Architectural Element Weights

As a means of preliminary screening, the CFA committee constructed a set of absolute and quantitative criteria for general military computer architecture. These criteria were compared quantitatively with the EW/ESM peformance of the relevant machines to determine the relevancy of the architectural elements to the requirements of EW/ESM. A complete description of the architectural criteria is given in Section III and will not be repeated here.

1. Absolute Criteria

These architectural characteristics defy quantification and can be used only to describe a qualitative characteristic of the computer. A computer either has or does not have floating point arithmetic capability. Table VII-6 was derived from the performance data for the nine computers evaluated by the CFA committee. A value was assigned to each characteristic based upon the computer architecture's performance. These values were used as a normalized basis for weighting the absolute architectural characteristics. An examination of the weights and architectural characteristics indicates, as might be expected, that very little difference exists in these criteria that could provide a basis for selection of an EW/ESM architecture. It is not surprising to find floating point arithmetic at a *slight* disadvantage, since it had been pointed out in previous sections that this capability is not frequently used in EW/ESM programs. However, to repeat, these absolute criteria display very few differences. They may be used in a screening process of vendors' candidate computers.

Table VII-5 — CFA/MCF Benchmark Tests

No.	Benchmark Description	Relevancy (EW/ESM)		
		Program S	Memory M	Processor R
1	Message Buffer and Transmission	−0.924	*−0.781	−0.333
11	Autocorrelation (Large)	*0.983	+0.220	+.315
8	Hash Table Search	−0.649	−0.614	−0.187
9	Linked List Insertion	−0.506	−0.662	+0.168
0	TTY Input Driver	−0.807	−0.267	−0.112
10	Presort (Large)	−0.332	+0.026	*+0.570
13	Boolean Matrix Transpose	−0.611	−0.197	−0.051
6	Target Tracking	−0.424	−0.342	+0.056
7	Digital Communicatons Proc	−0.539	−0.207	+0.028
12	Character Search	−0.254	−0.409	−0.006
14	Record Unpacking	−0.010	+0.199	+0.429
5	Array Manipulation	−0.325	−0.119	−0.081
2	Mulitple Priority Interrupt	−0.264	−0.248	−0.007
15	Vector-to-Scan Line Conversion	−0.454	−0.002	+0.010
4	Scale Vector Display	−0.125	+0.036	−0.051
3	Virtual Memory Exchange	+0.018	+0.072	+0.000

*Maximum relevancy

2. Quantitative Criteria

Architectural characteristics that do require description in terms of some quantification were identified as desirable military computer architectural characteristics by the CFA committee. These characteristics were also correlated with the EW/ESM performance of the computers analyzed by the committee. As noted in Section VII D.1, these architectural characteristics have been described and evaluated in previous sections and need not be repeated here.

Table VII-7 has been constructed to provide a further basis for the selection of computer/processors for EW/ESM.

E. Tool Availability Index

The ability to program a computer/processor efficiently is directly related to the software tools available to the programmer. After hardware architecture and microinstruction execution time, program efficiency becomes the most important aspect of computer/processor performance. The program determines the number of steps a computer/processor must take to accomplish a task. This is demonstrated in Appendix A where a microinstruction program is designed and stepped out for the Flywheel Tracker, a fairly simple, but demonstrative, algorithm.

Table VII-6 — EW/ESM Weights for Absolute Computer Arcitectural Criteria

Architectural Element	Value	Weight
1. Virtual Memory	1.909	1.027
2. Interrupts and Traps	1.913	1.025
3. Subsetability	1.949	1.006
4. Multiprocessor	1.949	1.006
5. I/O Controllability	1.949	1.006
6. Extensibility	1.949	1.006
7. Read-only Code	1.949	1.006
8. Protection	1.965	0.998
9. Floating Point	2.120	0.925

The "tools" to which we refer in this section are the means available to the programmer by which he constructs a program. These tools range from the familiar FORTRAN Compiler to the less familiar Text Case Design Advisors. Certain of these tools are basically necessary to the operation of the computer/processor. Certain of these tools are easily transferred in application from one architecture to the other, being essentially architecturally independent. Other tools are architecturally dependent. As mentioned previously, it is important that the software architecture, and TAI, at least, be independent of technology; such that computer/processor technology changes may proceed without the ponderous expense of "carrying" a software tool redesign ($26 to $28 million).

If one is to consider the case of EW/ESM alone, it is possible to speculate as to the applicability of some of these tools. Such was demonstrated in Section VI on *specification*, where TAI was addressed for its derived degree of contribution to EW/ESM performance. There can be no doubt that certain software tools are fundamentally necessary to EW/ESM programming performance. Others make a demonstrated, high contribution, while another category of tools contributes very little.

An examination of Table VII-8 indicates that there is almost no relevancy between the Tool Availability Index (TAI) and the performance of five architectures for which these data are available. Subsequently, in Section VIII, it will be shown that there is a high relevancy between software cost ($/instruction) and TAI. There appears to be little support for TAI as it relates to the S measure, described earlier in Section IV. It appears, therefore, that at least from the point of view of performance selection, TAI should be chosen at a minimum. This choice, as is demonstrated in Table VII-8, would result in a higher performance/TAI efficiency. From the point of view of performance, therefore, TAI selection should be minimized. The smaller the TAI, the higher the performance efficiency.

F. Physical Characteristics

The physical characteristics of a computer/processor may or may not be critical in the selection process. In a space system, weight, volume, and power are all important factors that require optimization. An airborne system requires optimization of weight and volume, with some relaxation on power. Most surface systems would only require some optimization of

Table VII-7 — EW/ESM Weights for Quantitative Computer Architectural Criteria Performance

	Architectural Element	Relevancy	Weight*	Units
M_2	Central Processor Transfer (min)	+0.601	0.158	Bits
B_1	Number of Computers Delivered	−0.535	0.128	Units
I	I/O control	+0.400	0.096	Bits
S_2	Central Processor State (min)	+0.375	0.090	Bits
J_2	Subroutine Linkage (min)	+0.373	0.089	Bits
L	Maximum Interrupt Latency	+0.274	0.066	Bits
B_2	Computer Inventory Value	−0.264	0.063	$M
U	Unassigned Instruction Space (Fraction)	−0.212	0.051	Fraction
D	Direct Instruction Address	−0.200	0.048	Bits
P_1	Physical Address Space Size	−0.186	0.045	Bits
V_1	Virtual Address Space Size	−0.162	0.039	Bits
J_1	Subroutine Linkage (max)	−0.152	0.036	Bits
V_2	Virtual Address Space Units	−0.126	0.030	Bits
S_1	Central Processor State (max)	−0.108	0.026	Bits
P_2	Physical Address Space Units	−0.093	0.022	Bits
M_1	Central Processor State (max)	+0.058	0.014	Bits
K	Virtualizability	0.000	0.000	Yes/No

*When all architectural elements are involved.

Table VII-8 — TAI Relevancy

Architecture	TAI (%)	Relevancy (r^2)			
		Performance		Efficiency P/T	
		EW	Gen-S	EW 0.018	Gen-S 0.015
AN/UYK-7	59	1.174	1.30	1.990	2.203
AN/UYK-19	42	2.108	0.93	5.019	2.214
AN/UYK-20	30	1.746	0.89	5.820	2.967
AN/GYK-12	21	1.525	1.14	7.262	5.429
AN/GYQ-21	63	2.293	0.82	3.640	1.302
		(Lower is better)		(Higher is better)	

volume, with weight and power noncritical. Current (1977) realizable weight, volume, and power curves are set forth in Section VI, based on airborne processor configurations that meet MIL-E-5400 specifications.

It would be natural that the selection of computer/processors on the basis of physical characteristics be weighted such that technology is rewarded by significant improvement over the current state of the art.

VIII. COST

While not directly involved in the technical performance of a computer/processor, cost is perhaps the most significant factor in procurement and can overwhelmingly dominate performance specification, evaluation, and second selection. As observed previously in architecture, the cost elements of a computer/processor are at least as complicated as the technical aspects. It is important that the purchaser have a good idea of the elements of his entire costs or the cost can rapidly escalate beyond his resources. The actual hardware is only one element that is presumed to include proper documentation. As previously implied, software considerations rapidly outgrow hardware costs. A new architecture may involve software tools on the order of $24 to $28 million. Estimates of the overall cost of software development and maintenance in the United States alone range from $15 to $25 billion. DOD spends an estimated $3 billion per year. Some 4 to 5 percent of the Air Force budget is in computer software, and 6 percent of NASA's budget is in computer software costs.

It is important to realize the magnitude of the above figures when a trade-off decision is to be made between hardware and software costs, where either is impacted by the other. A decision to use a new architecture with a resulting higher program cost can cause software costs to grow rapidly into billions of dollars, even through the cost per instruction may seem an insignificant percentage.

Summarized in the introduction was the cost per instruction for various architectural bases. This cost was significantly related to the total investment in an architectural base by

$$C_{\$/I} = 42.34\ B_2^{-0.149} \qquad \text{(VIII-1)}$$

$$r = 0.902$$

where

$C_{\$/I}$ = cost per instruction ($)

B_2 = total dollar value of architecture inventory ($M)

r = correlation confidence (1.0 = perfect) (see Fig. VIII-1).

The CFA Selection Committee generated the architectural attribute evaluation results that are the basic inputs to the cost computations [26]. The first of these are the data derived from test program experiments. These data indicate the relative efficiencies of the architectures in utilizing storage and processor hardware resources. The S measure is a count of the number of storage bytes required to contain programs, given an architecture. Differences in S can be directly related to differences in the amount of storage, and therefore, cost requirements. The M measure is a count of the number of bytes transferred between the CPU and main memory including cache) during execution of the test programs, and the R measure is a count of the number of bytes transferred among the registers of the CPU. M and R are clearly indicators of the hardware bandwidth requirements of an architecture to do a job. Everything else being equal, memory cost will be greater if more storage is required (larger S) or if the memory has to be faster (larger M or R). Similarly, the CPU cost will be larger if the processor has to be faster (larger M or R). The relationship between memory, CPU speed, and cost is taken as speed (in MIPS*) = $k \times \text{cost}^g$, where k and g are empirically derived constants.

*MIPS is millions of instructions per second.

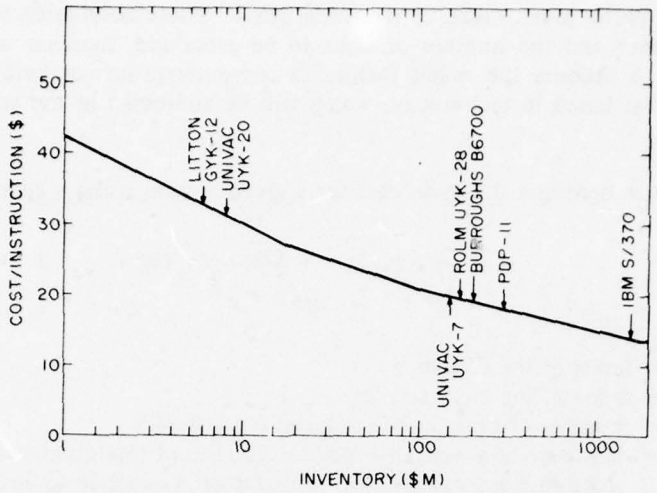


Fig. VIII-1—Instruction cost vs inventory investment

The other architecture attribute used in cost computations is the availability of software tools to aid in developing software for MCF systems. This was established by the Selection Committee by defining support software (i.e., compilers, editors, etc.) required for military applications and then evaluating the relative percentage of this support software available for each architecture. Using data from actual system developments, the Committee generated a curve relating this relative availability of software tools to the cost (per line of code) of developing software for an architecture. This data were ultimately used in the computation of life cycle software costs. [27]

The ensuing cost analysis is based upon CFA's "bottom-up" model, which predicts the cost of a particular computer architecture j, as applied to a particular military (EW or other) system i. It stands to reason that different architectures j will produce different results depending upon system requirements.

The computer resource life-cycle cost for system i and architecture j is defined as

$$C_{ij} \equiv HW_{ij} + ASw_{ij}, \qquad \text{(VIII-2)}$$

where

HW_{ij} = hardware life cycle cost

ASw_{ij} = applications software life cycle cost.

A. Hardware

As may have been implied in the foregoing, the cost of computer/processor hardware is dependent upon several elements; processor speed, main memory capacity, and secondary

memory capacity pretty much make up the initial costs. These costs must be added to lifetime cost of maintenance and the number of units to be processed, together with the number of units produced, to become the major factors in computer/processor hardware costs. These costs are somewhat based in technology, which will be addressed in the subsequent and final Section IX. [26]

The computer hardware life-cycle cost for a given system using a specific CFA is defined as

$$HW_{ij} = n_i L_h (P_{i-j} + MM_{i-j} + SM_{ij}), \tag{VIII-3}$$

where

i = index of the system
j = index of the architecture
n_j = number of units to be produced for system i
L_h = hardware life-cycle cost factor, i.e., ratio of total hardware life-cycle cost to hardware acquisition cost. This factor is assumed to be 2 for a 10-year life cycle
P_{ij} = processor acquisition cost
MM_{ij} = main memory acquisition cost (see Fig. VIII-2)
SM_{ij} = secondary memory acquisition cost (see Fig. VIII-2).

1. Processor Speed Costs

Principally based in technology, the processor speed as has been demonstrated is a factor of memory speed. At present (1975-1977) there are three technologies that dictate three ranges of memory access time; core, MOS, and bipolar, at corresponding costs ranging from 0.1 cents/bit to 1.0 cents/bit. These technologies represent memory access time (and processor speeds) of from 300,000 IPS to 80 MIPS, (see Fig. VIII-3). The range of values is not obvious, since there is a give and take in memory capacity involved.

Tutorially, the initial acquisition cost of a processor with $\bar{\tau}_i$ average instruction processing time is

$$C_{Hi} = \frac{k}{\bar{\tau}_p^{g_n}} \tag{VIII-4}$$

where

C_{Hi} = cost of the ith processor
k = 6.3×10^4
g_n = 0.4
τ_p = average instruction execution time (in μsec).

The processor acquisition cost for system i using architecture j is defined as

$$P_{ij} = K(a_{ij} Mr_i)^{0.4} \tag{VIII-5}$$

where

k = constant relating to processor cost
a_{ij} = processor speed ratio
Mr_i = operating speed in millions of instructions per second (MIPS).

Equation (VIII-3) follows from a commonly cited relationship between performance and cost, namely performance = constant × cost^g. For the purpose of the bottom-up (BU) model, g was assumed to be 2.5. To obtain a value for k, in Eq. (VIII-5), CFA used the fact that recent cost/speed data for several military processors seem to indicate that speeds of 0.5 MIPS ($\tau_p = 2\ \mu s$) and processor costs of $48,000 are representative values; consequently,

$$K = 48 \times 10^3/(0.5)^{0.4} = 6.3 \times 10^4.$$

This value of K is used in subsequent calculations for 1976 processor cost estimates and is reduced by a factor of 10 for 1985 processor cost estimates based on an assessment of hardware cost reduction over the next decade (see also Section IX.D).

2. Main Memory Costs

At this time, there are three types of main memory technologies — core, MOS, and bipolar — which represent three distinct characteristics in access time, capacity, and cost.

The main memory acquisition cost for system i using architecture j is defined as

$$MM_{ij} = c_b(b_{ij}PM_i + DM_i), \tag{VIII-6}$$

where

MM_{ij} = main memory acquisition cost (in dollars)

b_{ij} = static storage ratio

PM_i = main memory (in bits) required for program storage in system iM_i is derived from system requirements; P is estimated fraction of M_i dedicated to program storage vs data storage.

DM_i = main memory (in bits) required for data storage in system i; M_i is derived from system requirements; D is estimated fraction of M_i dedicated to data storage vs program storage.

c_b = cost per bit of main memory. Examination of the price per bit of recent militarized memory systems indicates an average cost of 4 cents per bit; i.e., $5000 per 16,000-byte memory module. This value is used in 1976 cost estimates; 0.4 cents is assumed in 1985 cost estimates.

(See also Figs. VIII-2 and Fig. VIII-3).

3. Secondary Memory Costs

Inherently characterized by slower access time and enormous storage capacity, secondary memories are important adjuncts to the computer/processor. Naturally, secondary memories cost less (see Fig. VIII-2).

The secondary memory acquisition cost for system i using architecture j is defined as

$$SM_{ij} = C_a(b_{ij}P'Ma_i + D'\ Ma_i) \tag{VIII-7}$$

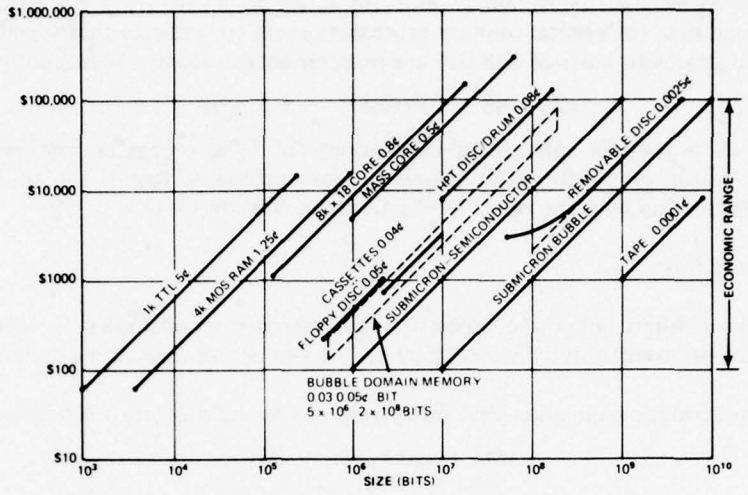


Fig. VIII-2—Cost vs memory size. Copyright © 1977 by Cahners Publishing Co., Inc. Reprinted, by permission, from *EDN* Vol. 22 (No. 9), pp. 21-24 (May 1977).

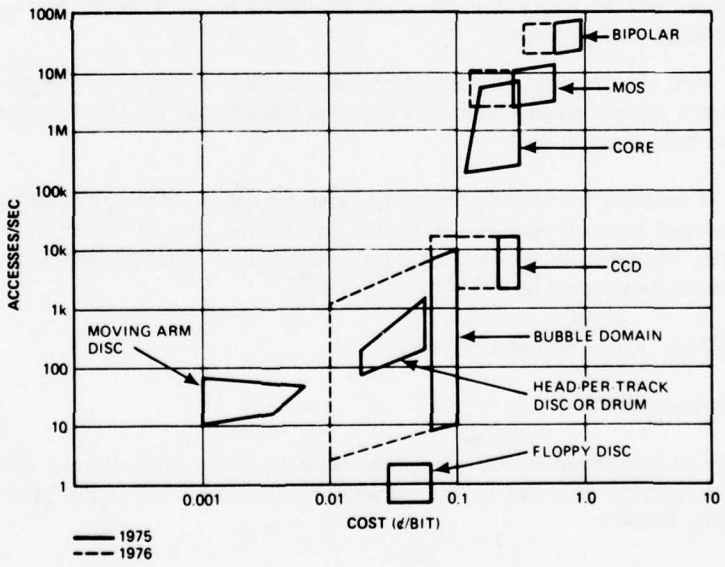


Fig. VIII-3—Access time vs cost (1975-1976). Copyright © 1977 by Cahners Publishing Co., Inc. Reprinted, by permission, from *EDN* Vol. 22 (No. 9), pp. 21-24 (May 1977).

where

SM_{ij} = secondary memory acquisition cost (in dollars)

b_{ij} = static storage ratio

$P'Ma_i$ = secondary memory (in bits) required for program storage in system i; Ma_i is derived from system requirements, whereas P' is the estimated fraction of Ma_i used for program storage vs data storage.

$D'Ma_i$ = secondary memory (in bits) required for data storage in system i. Ma_i is derived from system requirements, whereas D' is the estimated fraction of secondary memory used for data storage vs program storage.

C_a = cost per bit of secondary memory.

(See Figs. VIII-2 and -3.)

Examination of the price per bit of current militarized disc systems indicates an average cost of 0.2 cents per bit, e.g., a 36-Mbit disc system costs $72,000. This value is used in 1976 cost estimates; a cost reduction of 10:1 in the next 10 years is assumed, so a price of 0.02 cents per bit is used in 1985 cost estimates (see also Section IX.D).

4. *S,M,R Costs*

The processor speed ratio a_{ij} and static storage ratio b_{ij} attempt to capture the ability of the jth architecture relative to system i. They are derived by measuring the performance of the three architectures on Benchmark test programs and by estimating the relative importance, or weight, of each program to computations characteristic of each system (see Table VII-5). The performance of each architecture on the test programs is summarized in what are called the S, M, and R measures, where

S_{kj} is a measure of the amount of memory (in 8-bit bytes) needed to represent test program k on architecture j.

M_{kj} is a measure of the processor/memory transfers required to execute test program k when using architecture j.

R_{kj} is a measure (in 8-bit bytes) of the number of internal register-to-register transfers required by the processor to execute test program k on architecture j.

See Section IV for details of the test program experiment.

The relevancy of the kth test program to the ith system is given by factors W_{ik}, which were obtained by first dividing the Benchmark programs into two categories: programs that relate principally to I/O, and those that are associated with traditional processor/memory functions. Within these categories, varying degrees of functional overlay occur among the test programs. Initially a gross value was estimated for each category; subsequently, this value was distributed across the programs of the category. As an example of how the weights were distributed, see Section VII.

The processor speed ratio a_{ij} and the static storage ratio b_{ij} were obtained by combining the above quantities in the following manner.

$$a_{ij} = \frac{3 \sum_{k=1}^{12} W_{ik}(3M_{kj} + R_{kj})}{\sum_{m=1}^{3} \sum_{k=1}^{12} W_{ik}(3M_{km} + R_{km})} \tag{VIII-8}$$

and

$$b_{ij} = \frac{3 \sum_{k=1}^{12} W_{ik} S_{kj}}{\sum_{m=1}^{3} \sum_{k=1}^{12} W_{ik} S_{km}}. \tag{VIII-9}$$

B. Software

As implied earlier, software costs may outweigh hardware costs when viewed in the total life-cycle cost picture. Figure VIII-1 showed a strong correlation between total dollars inventory investment and the cost-per-instruction for software. This correlation is even stronger when cost per instruction is related to the TAI:

$$C_{si} = 1138\, T_{si}^{-1.07} \tag{VIII-10}$$

$$r = 0.99$$

where

TAI = Tool Availability Index (%)
r = correlation confidence

and $\pm\ \sigma$ is

$$781\ (TAI)^{-1.07} < C_{si} < 1984\, TAI^{-1.07}. \tag{VIII-11}$$

However, since TAI is not as easily evaluated as investment inventory, the relationship of Fig. VIII-1 is more usefully applied.

Some note of caution should be brought to the attention of the EW community, here. In Sections V, VI.B, and VII.E, there appears to be very little relevant influence of TAI on EW/ESM performance, so it may be possible to tolerate poorer TAIs and, consequently, high program costs and still experience more cost-efficient computer/processors. In other words, it should be noted that architectures with lower TAI do come out with higher cost efficiencies in total life-cycle costs (LCC) (see Table VIII-1).

The applications software life-cycle cost for system i using architecture j is defined as [26,27]

$$ASw_{ij} = C_{sj} S_i L_s, \tag{VIII-12}$$

where

C_{sj} = cost (in dollars) per instruction of applications software for architecture j
S_i = applications software size (in instructions), derived from the system proponents data (Table II-3 and Fig. II-6)
L_s = applications software life-cycle cost factor, i.e., ratio of applications software life-cycle cost to initial acquisition cost (= 5.5).

Table VIII-1 — EW/ESM Cost Efficiency

Architecture	S	Normalized C_{sj}	C_{sj} ($k)	Throughtput Cost ($/pulse)
IBM 370	0.82	1.434	206.93	070.12
PDP-11	0.89	0.738	393.63	116.05
UYK-28	0.93	1.080	324.65	092.95
UYK-7	1.14	0.653	648.79	093.45
UYK-20	1.30	2.114	349.52	082.90

1. TAI Cost

The MCF provides an evaluation of the costs of each element in the previously described TAI elements. These costs, together with the required TAI elements for each architecture, became the basis for the indices used in this report. Table VIII-2 presents the estimated value of these TAI elements, together with their EW/ESM weights.

The total dollar value of TAI was plotted in Fig. VIII-4 for representative architectures and is presented in Table VIII-3. There appears to be a high confidence relationship between TAI and costs:

$$C_{TAI} = 244\ TAI^{1.009} \qquad \text{(VIII-13)}$$

where C_{TAI} = total software tool cost ($K)

TAI = Tool Availability Index (%)

r = correlation confidence, 0.994.

C. Life Cycle

Table VIII-4 summarizes the total life-cycle cost (per unit) for the five architectures for which the most current data are available. This table (VIII-4) should be tempered by Table VIII-1, in which recognition was given to the EW/ESM throughput performance of the architectures and the relatively low (EW/ESM) relevancy to TAI. It should be noted that the AN/UYK-20 rapidly jumps to second place from last (Table VIII-4) when it is applied to EW/ESM in cost performance. This is only to emphasize the importance of the interplay between architecture j and system i. In other applications or with architectures that have not been addressed, the best architecture for the best system can change again significantly.

It should be noted that the CFA/MCF work described above is for general military applications. It does not rule out the relative advantages of special applications such as EW/ESM that make some architectures (such as the AN/UYK-20 with its relatively low, 30% TAI), a good EW/ESM (cost) choice.

Table VIII-2 — Software Tool Costs

EW Weight (Higher is Better)	Software Tool	Cost (1977)
1.61	Noninteractive Debugging Aids (CMS-2)	$50k
1.61	Language-Independent Monitor	210k
1.34	*Compiler + Cross-Compiler (CMS-2)	3300k
1.12	Real-Time + Time-Sharing Operational System	3500k
1.07	*Computers + Cross-Compilers (TACPOL)	1000k
1.07	*Test Case Instrumenter + Analyzers (TACPOL)	280k
1.06	*Test Case Instrumenter + Analyzers (CMS-2)	280k
1.07	*Noninteractive Debugging Aids (TACPOL)	50k
1.06	Language-Dependent Monitor (Assembler)	50k
1.00	Assembler	135k
1.00	Macro Assembler	800k
1.00	Basic Linker	130k
1.00	Simple Overlay Linker	210k
1.00	*Interactive Debugging Aid (Assembler)	300k
1.00	Test Data Auditor	140k
1.00	Test Data Generator	350k
1.00	Integrated Library	100k
1.00	Text Processing System	630k
1.00	Interactive Source Editor	130k
1.00	Batch Source Editor	100k
1.00	Data Base Management System	4200k
1.00	General Purpose System Simulator	700k
0.99	Instruction Simulator	350k
0.64	*Reformatters (Fortran)	110k
0.64	*Test Case Instrumenters + Analyzers (Fortran)	280k
0.64	*Compilers + Cross-Compilers (Fortran)	1000k
0.63	*Noninteractive Debugging Aids (Fortran)	50k
0.63	Extended Overlay Linker	500k

*Alternative choices represented

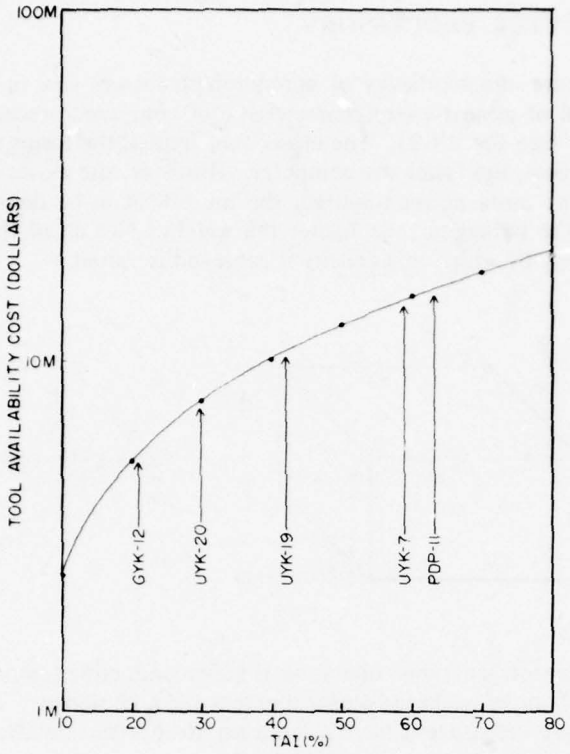


Fig. VIII-4—Tool availability cost

Table VIII-3 — TAI Costs

TAI (%)	Cost ($)	Architecture
21	$5,075k	AN/GYK-12
30	8,035k	AN/UYK-20
42	10,295k	AN/UYK-19
59	15,755k	AN/UYK-7
63	15,495k	AN/GYK-21

Table VIII-4 — EW/ESM Architecture Life-Cycle Costs

Architecture	$t_p(\mu s)$	B_2($M)	Chi($k)	C_{sy}($k)	10yr LCC($k)
IBM S/370	2.182	16,000	46.1	152.6	645.3
PDP-11	2.293	311	45.2	274.5	935.1
AN/UYK-28	2.108	169	46.8	300.6	1,007.8
AN/UYK-7	1.174	147	59.1	306.9	1,096.7
AN/UYK-20	1.746	8	50.4	473.6	1,448.2

N.B. Based on previously noted program structure.

IX. FUTURE COMPUTER/PROCESSORS

The key to future characteristics of computer/processors lies in technological research today. The sum total of almost every characteristic of computer/processors lies in the circuit density of technology (see Fig. IX-1). The closer the circuits, the faster a signal propagates, the faster the memory access, the faster the computer. However, the closer the circuits, the higher the energy density, the more power required, the more heat to be dissipated. The higher the density, the smaller the packaging, the lighter the weight. Not all of this is directly linear, of course. The technology by which this density is achieved is varied.

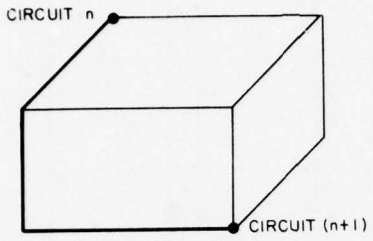


Fig. IX-1—Circuit density

The exotic Josephson junctions, operating at superconducting temperatures, have no resistance; therefore, they can be packaged closer together for high density. However, the materials by which J^2 circuits are constructed have a tendency to disassociate after a relatively few (10) cycles of temperature change.

Emitter Cathode-Follows Logic (ECL) devices realize high propagation times but must be "forced" through high relative power. This creates heat dissipation problems and accompanying reliability problems.

While light source lithography permits use of impressive circuit densities, a limit has been reached due to the diffraction resolution limits of the source. The relativistic x-ray and electron beam technologies are now being pursued to achieve even higher density masking through the higher diffraction resolution of these sources. However, these higher circuit densities, using familiar techniques and materials, are accompanied by slower access times in an almost linear, inverse relationship (see Fig. IX-2). The picojoule line is the barrier that is stopping most technologies, regardless of ability to achieve high VSLI packing densities through E-beam or x-ray diffraction techniques. Here, again, Josephson junctions may provide the breakthrough of the "picojoule barrier."

A. Statistical Forecasting

It is apparent that the fitting of trends merely describes the movements of a series and that this type of work belongs to the domain of descriptive statistics. Having found the equation of a trend, the obvious thing that suggests itself is to extrapolate or to estimate a value that lies beyond the range of values on the basis of which an equation was originally obtained. However, its success depends on many factors.

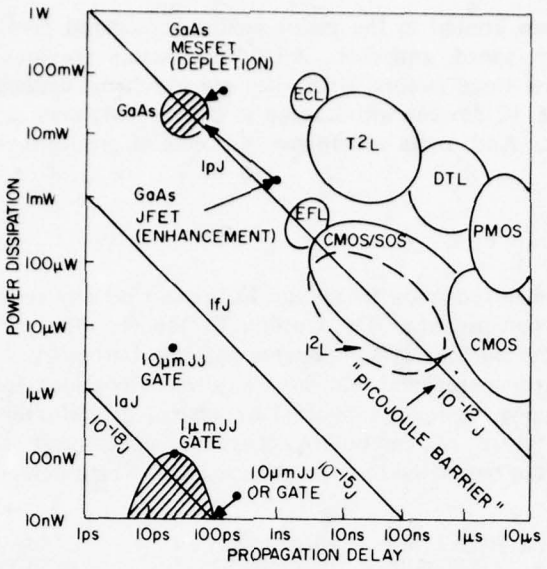


Fig. IX-2—The "picojoule barrier" power dissipation vs propagation delay for various logic technologies. Boundaries exist at 10^{-21} J for room temperatures and at 5×10^{-23} J for 4 kelvins.

The basic question almost always asked is whether the forces that have operated in the past will continue to operate in the future and to operate in the same way. It could be a starting point, a base from which to proceed to a final prediction.

Statistics, in its current state of development, comes somewhere between being of relatively little value and providing a complete solution of the problem of forecasting. When used intelligently with appreciation of its possibilities and recognition of its limitations, statistics provides invaluable quantitative aids in reaching well-informed and wise decisions.

In the ensuing forecasts projected in this section, two factors are employed for confidence. One is an overwhelming industrial inertia that propagates the vast IC market such that comparable massive (technological) power would be required to shift the predictions indicated. In other words, the factors predicted are so massive that they will continue as projected by the curves. Second, there is technology that can support these predictions, even though they are not fully developed. Finally, commercial systems, and subsequently military systems, lead technology by a significant number of years (commercial leads military by roughly seventeen years). The reason for this is that initially industry must adjust to new technology and military must adjust to new commercial products that are not as rigid in specification as military requirements. Therefore, the forecasting here is not as risky as may appear. By examining new technology results alone, it would be possible to predict commercial and subsequently military computer/processor performance in the next twenty years. Combining the support of new technology with the events of the past twenty years makes it possible to add more confidence to the predictions.

The forecasts are limited to the major factors considered pivotal to computer/processor development: density, speed, and cost. All of the factors previously discussed will develop parasitically with these three factors, since they are somewhat dependent upon the same technology. For example, IC devices will develop in a systematic manner to accommodate increasing processor speeds. And, costs will follow the costs of producing ever-decreasing processor costs.

B. Density

As noted in the introduction to Section IX, circuit density is pivotal to the prediction of computer/processor performance. Dr. Gordon E. Moore, then of Fairchild Semiconductor, noted in 1964 that the number of components per chip (circuit) had doubled every year since 1959. Dr. Moore further predicted that this trend would continue (see Fig. IX-3). This curve appears to be continuing its trend, supported by technology. Moore's curve, then, is the basis of predicting the future of computer/processors, since most all characteristics of the computer/processor can be related to circuit density with a high degree of confidence.

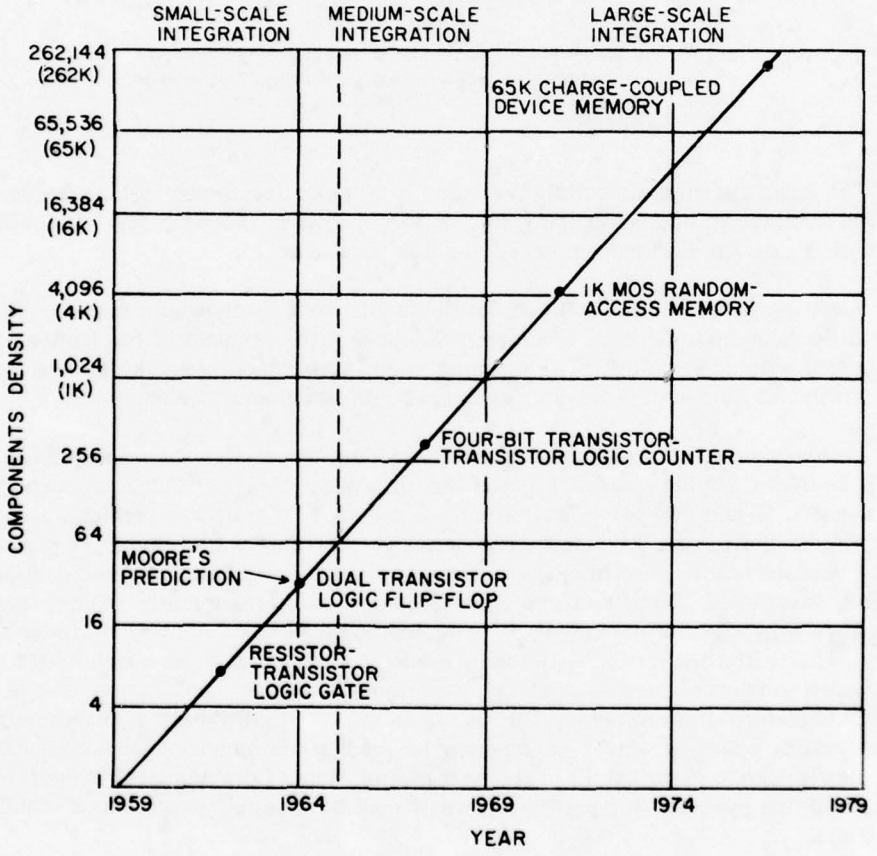


Fig. IX-3—Moore's curve

It takes about three years between initial design and the production of microprocessor designs. Any VLSI bus implementation would be foolish unless at the start of a design it is possible to forecast the manufacturable logic density that will be available some years ahead. [29]

Fortunately, progress in LSI technology over the past ten years has been very consistent. Considering the progress on only the fastest moving technology—metal-oxide semiconductors (MOS)—we see that the speed × power product has decreased by two orders of magnitude while gate-propagation delay has decreased by almost an order of magnitude. Gate density and practical chip size for high-volume production also have increased steadily. On that basis, it is possible to predict with a fair degree of certainty what kind of complexity will be used five to ten years from now. Today, we can put 8000 gates on a chip to implement random logic—more, of course, in the repetitive designs of memories and particularly of read-only memories. Our predictions suggest that by 1981 those random logic designs will have a complexity equivalent to about 20,000 gates on a single chip.

Because of recent advances in electron-beam and x-ray lithography, coupled with better understanding of small geometry devices, there is no reason to think that in the next decade the rate of increase in complexity will depart appreciably from the one shown in Figs. IX-4 and IX-5.

It is obvious that the technologcal acceleration shown in Figs. IX-4, IX-5, and IX-6 cannot continue unabated forever. However, as long as we are several orders of magnitude away from the theoretical limit, progress is likely to continue at the rate shown. A gross estimate of a practical limit for MOS technology is a circuit using complementary MOS (CMOS) technology,

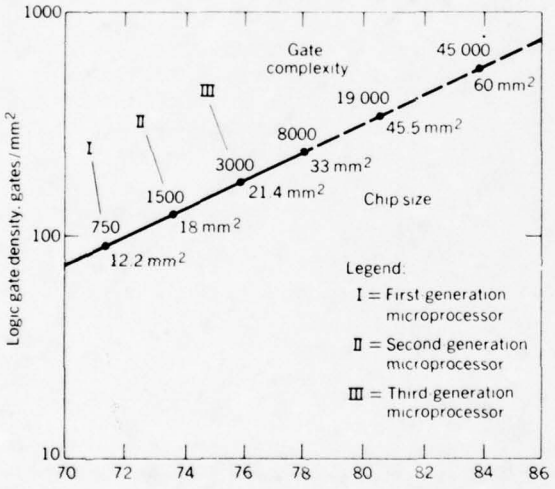


Fig. IX-4—Logic gate density trend. Copyright © 1978 by Institute of Electrical and Electronic Engineers, Inc. Reprinted, by permission, from *IEEE Spectrum* vol. 15 (No. 5), pp. 28-31 (May 1978). Author's permission also obtained.

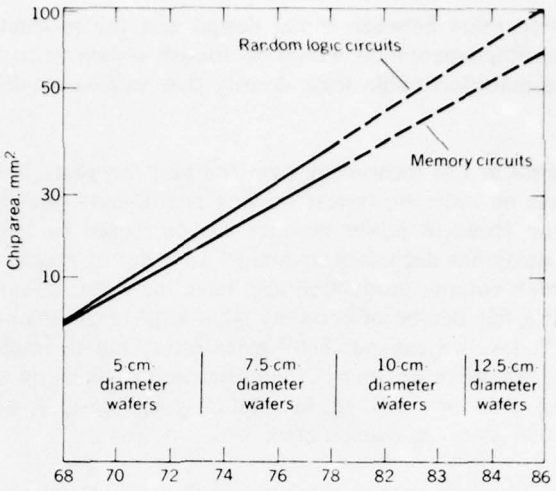


Fig. IX-5—Projection of logic and memory circuit sizes. Copyright © 1978 by Institute of Electrical and Electronic Engineers, Inc. Reprinted, by permission, from *IEEE Spectrum* vol. 15 (No. 5), pp. 28-31 (May 1978). Author's permission also obtained.

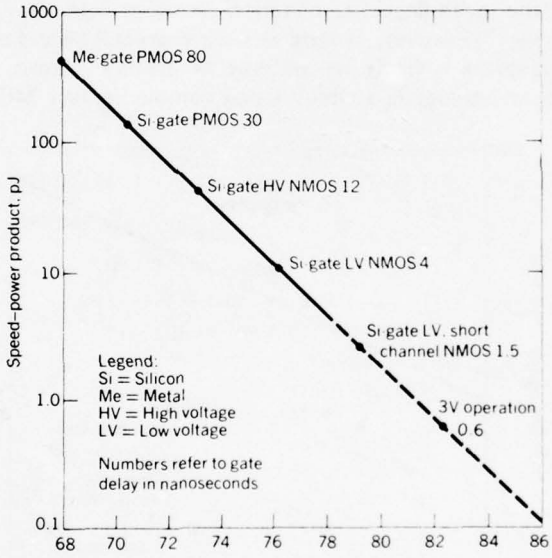


Fig. IX-6—Speed-vs-power trend of LSI production. Copyright © 1978 by Institute of Electrical and Electronic Engineers, Inc. Reprinted, by permission, from *IEEE Spectrum* vol. 15 (No. 5), pp. 28-31 (May 1978). Author's permission also obtained.

operating at a supply voltage of 400 mV, having a minimum line width of 0.25 μm, dissipating 1 W at an operating frequency of 100 MHz, having a chip size of about 5 × 5 cm^2, and having the complexity of about 100 million gates! The trends shown in Figs. IX-4, IX-5, and IX-6 are still very far from a practical limit; therefore, technological acceleration will continue beyond the next decade.

In only seven years the industry has advanced from simple-minded microcomputers, remarkable less for what they could do than for what they suggested could be done, to the microcomputers that rival high-end minicomputers in performance.

It is Faggin's [29] assessment that future processor development will take place along two paths: One path will be essentially the same one that technology has been following for the last twenty years—putting more and more functions on a chip and making the chip run faster and faster. An example is integrating onto a single chip the CPU, memory, and I/O circuitry of an older design that previously required separate chips. This path will continue to be plagued by the need for compatibility with prior designs. As a result, architectural progress along this path will be relatively slow.

The other path, that of developing a parallel-processor architecture, means taking a fresh look at new technology and at the many new processor applications that have recently emerged. Given the fact that a complex CPU or even a full single-chip microcomputer can be purchased for a few dollars, it seems appropriate to devise a structure using a multiplicity of microcomputers providing more processing power than would be possible or practical using a single CPU with traditional architecture.

The previously mentioned, the "picojoule barrier" does show a predictable improvement as displayed in Fig. IX-6. This tends to belie the "barrier" concept if voltages can continue to decrease without noise probems. Such a trend would result in logic densities as depicted in Fig. IX-4.

1. Instruction Timing

The improvement of instruction timing, or inversely, the number of operations per second that an architecture can expect, is related to the number of circuits by

$$\frac{1}{t_i} = 0.115\ N_c^{0.488}$$

$$r^2 = 0.948, \qquad \text{(IX-1)}$$

where

t_i = time to execute average instruction (μs)

N_c = number of circuits

r^2 = correlation coefficient.

Obviously, as the density increases, the processing capability for a given volume or weight improves, as indicated by the trends in (ultimately) Moore's curve.

2. Volume

The size of a computer/processor for a particular throughput must improve almost directly with the improvement in the speed vs power curve. This relationship follows the relation

$$V = 239\ N_c^{0.851}$$
$$r^2 = 0.993, \qquad \text{(IX-2)}$$

where

V = volume in cubic inches

N_c = number of circuits

r^2 = correlation coefficient.

The improvement must be in speed, power, or as the number of circuits increases in density, the speed of these circuits will decrease as an indirect function (see Figs. IX-2 and IX-6).

3. Weight

The weight of computer/processors also shows a high correlation to the number of circuits active in the architecture. Of course, as densities improve, so will the weight of computer/processors for a particular performance. At present, weight is related to the number of circuits by

$$W_t = 20.284\ N_c^{0.531}$$
$$r^2 = 0.972, \qquad \text{(IX-3)}$$

where

W_t = weight in pounds

N_c = number of circuits

r^2 = correlation coefficient.

This, of course, will not remain the same; as the number of circuits per chip increases, weight will remain constant but throughput (MIPS $= 1/t_t$) will improve with circuit density.

4. Cost

Cost, which is to be covered subsequently in this section as a major topic, is related to the number of circuits by

$$C_{hw} = 594.74 N_c^{0.795} \qquad \text{(IX-4)}$$
$$r^2 = 0.965,$$

where

C_w = cost of hardware ($)

N_c = number of circuits

r^2 = correlation coefficient.

Naturally, again, as the cost to produce these chips goes down, the cost of performance (MIPS) will decrease also (see Fig. IX-7).

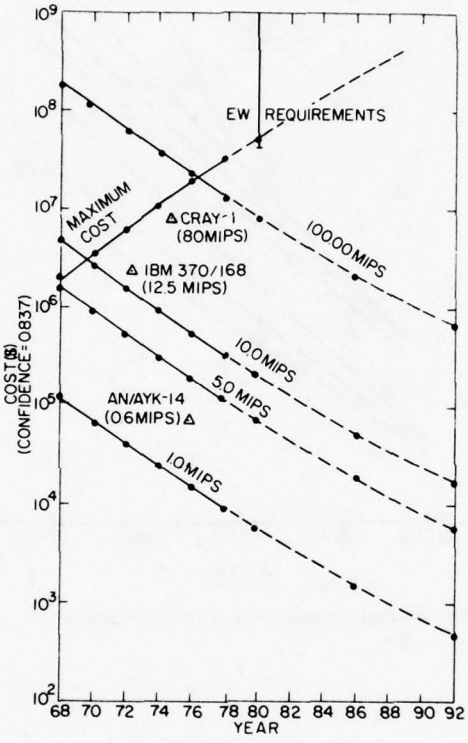


Fig. IX-7—Projection of computer/processor cost (varied throughput)

5. Memory

Ultimately, the use of magnetic tape for the recording and storage of information in large on-line systems will probably be totally phased out. Very high-density magnetic discs will be in common use, but newer developments also can be anticipated. For example, the use of large semiconductor random access memories may be standard in the future. Developments in the area of magnetic bubbles will also lead to major breakthroughs, at least in the magnetic area, providing larger and larger random access storage systems. [30]

Figure IX-8 presents a projection of the capacities of magnetic discs and bubble systems for the year 2000. Note that at the middle upper portion of the display, one point has been added. This is the point that represents the current available storage density for the video disc, which is currently being developed by several different organizations. Its potential for the year 2000, as of this point in time, has not been assessed although it can be expected to grow with a minimum rate of increase equal to that of magnetic systems. The ability to store 10^9, 10^{10}, or 10^{11} bits in a square inch certainly opens up whole new applications.

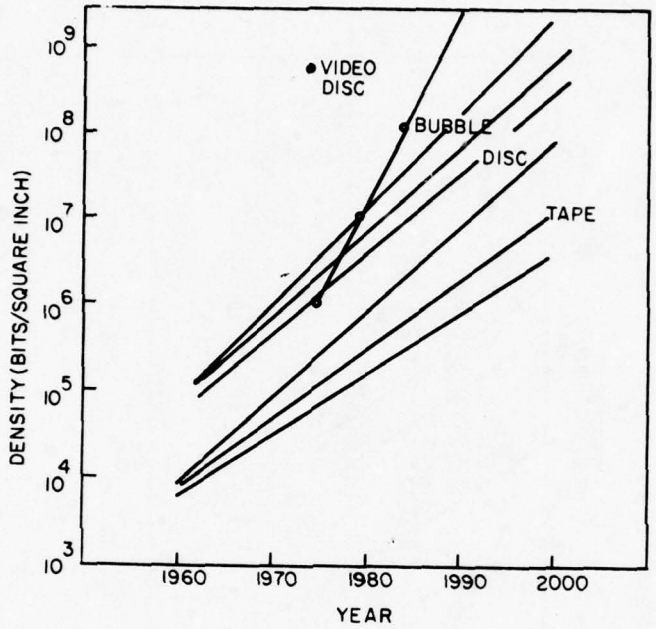


Fig. IX-8—Projected densities of storage devices

6. Military

Figure IX-9 summarizes the foregoing discussion as it impacts military processors. Using technology projected for logic gate densities as the trend-setter, it is possible with some degree of confidence to project the memory density of military computers. It should be noted that military performance is not a smooth, linear curve but advances in steps with a delay of about 17 years behind commercial performance. The inverted function depicted in Fig. IX-9 is a direct result of advancing technology in circuit densities. As technology permits the production of higher density circuits, the volume (size) of military computer/processors will decrease for each megabit of memory capacity. As a benchmark, the emerging AN/AYK-14 was used for comparison.

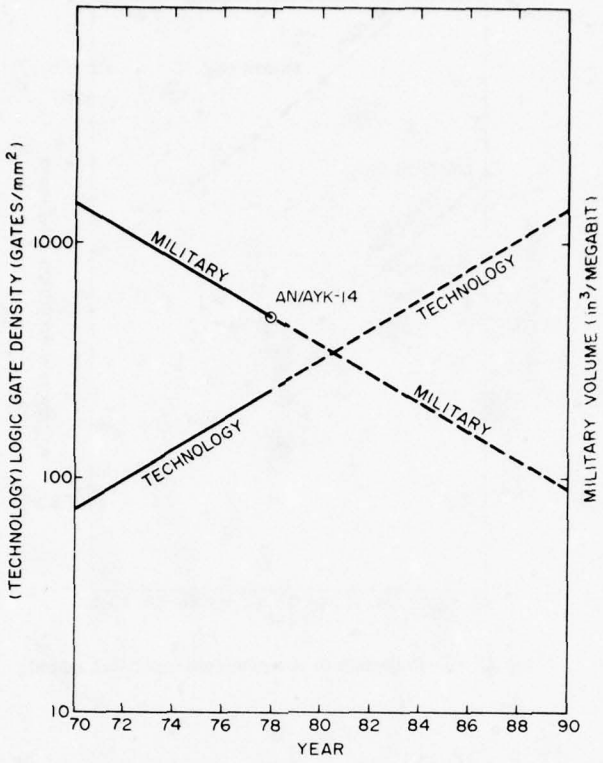


Fig. IX-9—Projection of computer/processor densities

C. Throughput

The principal criterion of EW/Military computer/processor evaluation in this report has been throughput (or speed). It is natural, therefore, to project the performance of computer/processors from the throughput point of view. Figure IX-10 uses the speed-power projection of Fig. IX-4 as the criterion for projecting military computer/processor throughput performance. Again, using the evaluated performance of the emerging AN/AYK-14 as a benchmark of military (EW) computer/processor performance, a predictor curve was drawn in Fig. IX-10. This curve projects the EW weighted average instruction time (in μs) as essentially the reciprocal of a computer/processor's throughput.

As another point of reference, the range of required performance (ca. 1978) from Fig. II-1 is depicted in Fig. IX-10 for platforms from 10,000 to 70,000 ft altitude (269 to 3443 MIPS). This performance requirement, of course, is for 100% coverage. The immediate impact of Fig. IX-10 is that at present, military computer/processors can only hope to achieve 0.22% of total throughput for an EW environment (real time) and projects to 100% (at 10,000 ft) in 1991 military technology — that is, presuming that the EW requirement does not increase in the meantime. At present, projected EW requirements do not appear to increase as fast as the projected improvement of computer/processors.

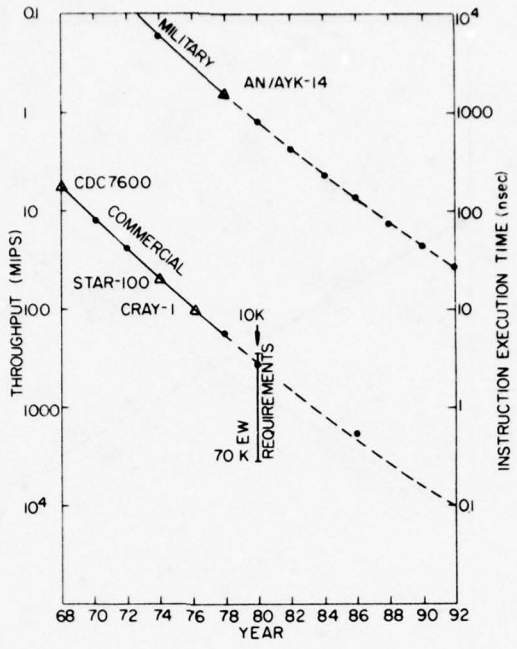


Fig. IX-10—Projection of computer/processor throughput

1. Volume

In the previous section, computer volume was related to circuit density by Eq. (IX-2). The number of circuit elements was also found to correlate highly (0.948) with throughput (in MIPS), by Eq. (IX-1). When these two relationships are used together with the projection of circuit densities from Figs. IX-4 and IX-5, one obtains a fairly accurate basis of projecting gross computer characteristics such as volume.

Depicted in Fig. IX-11 is a projection of computer/processor volumes for various throughput values (1, 10, 100 MIPS) through the year 1992. These projections follow the familiar methods of statistical forcasting described earlier and are based upon the rather well-behaved technology characteristics of the previously mentioned Figs. IX-4 and IX-5 and Moore's curve (Fig. IX-3). To illustrate the degree of confidence in such curves, some architectural benchmarks are placed in the figure. It should be noted that the military AN/AYK-14 appears to register rather well with the curves of volume, where volume is an important characteristic to optimize for airborne computer application. The maximum volume line may require some explanation. This curve is based upon computer configurations that represent the maximum capability configuration produced on an analytical basis.

As may be expected, there is a general (almost linear) downward trend of performance per volume in the future, although full 100% real-time EW requirements remain a rather taxing requirement for the computer/processor. This "EW volume" requirement is based upon the volume/throughput relationship previously developed.

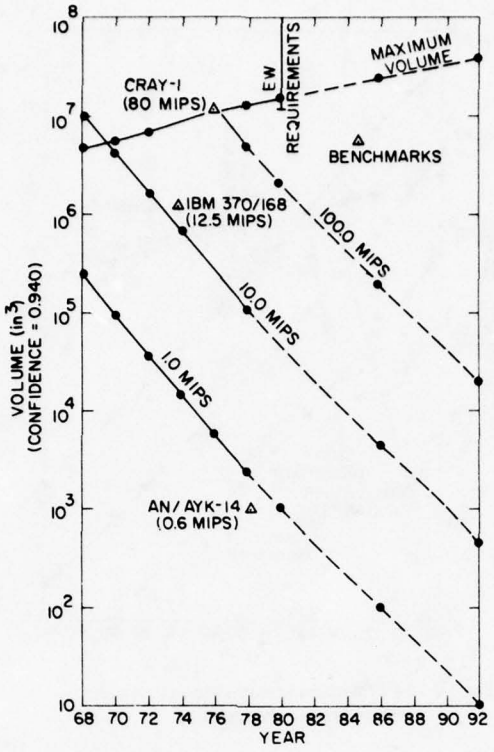


Fig. IX-11—Projection of computer/processor volume (for varied throughput)

2. Weight

As developed previously for Section IX.C.1, *Volume*, the projection of computer/processor weight in the future (1992), is based primarily upon throughput and circuit density arguments. Weight was related to circuit density in Eq. (IX-3). The remaining relationships of throughput and yearly improvement of circuit density are as previously developed.

Figure IX-12 depicts the results of these relationships graphically, as was done for the volume relationships. There are no more particularily surprising projections for computer/processor weights through 1992. The confidence index noted on the ordinate at left indicates fairly high confidence (0.921) that such goals will be realized. This confidence index is based upon the classical statistical, correlation coefficient explained in relation to Eq. (VII-1).

It should be noted that the impetus for these predictor curves depends upon some ongoing normal level of R&D in addition to market demands. It is not sufficient to say that these goals will come about without the R&D effort. On the other hand, there is always a chance that exceptional R&D (breakthroughs) can result in exceeding these goals. However, the improvement projections are being based on the anticipated breakthroughs.

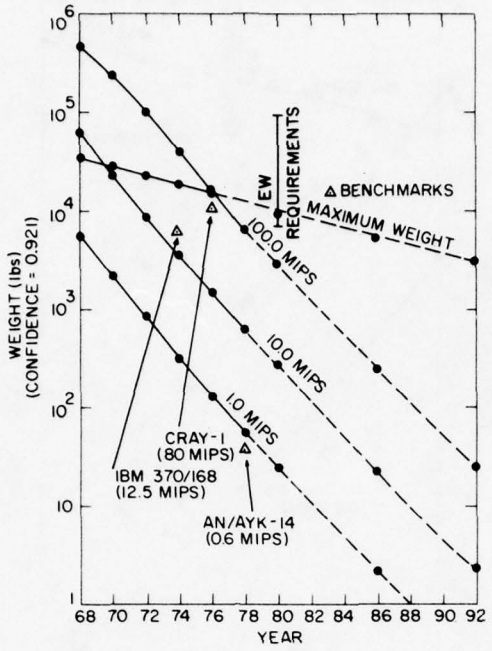


Fig. IX-12—Projection of computer/processor weight (varied throughput)

D. Cost

In Section IX.A, the instruction timing of a computer/processor (or inversely, the throughput) was shown to have a high correlation (0.948) with the number of logic chips in the processor, Eq. (IX-1). Further, as may be expected, the cost of the processor displayed an equally high correlation (0.965) with the number of logic chips in the processor. It would stand to reason, then, that processor cost can be related to throughput with relations (IX-1) and (IX-4).

Again, the same methodology is pursued as for throughput, volume, and weight; however, to relate the above relationships to cost projection, a new prediction has to be applied. The predictor deviates from the Moore's curve, speed vs power projection. The curve depicted in Fig. IX-13 is the trend of silicon-device prices that dominate the costs of circuits dominating computer/processors. Again arguments that other technology and other materials can influence a projection based upon silicon devices are countered by the competitive market that the silicon-device industry creates for such other devices; and vice-versa.

The curve of Fig. IX-13 is used as the basis of trend prediction for the cost projections of Fig. IX-7. Here, as was mentioned, throughput and number of circuits were used to relate to the (silicon device) trend-predictor curve in such a manner as to project the cost trends of computer/processors. Finally, certain well-established architecture configuration costs are used to link total commercial costs to the trends of silicon devices. Significantly, the AN/AYK-14

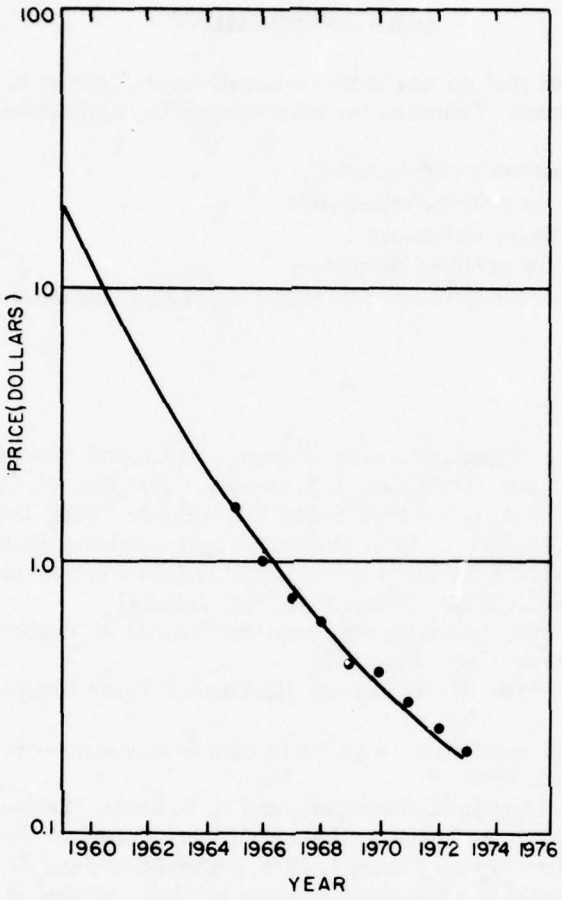


Fig. IX-13—History of silicon device prices

benchmark displays a large deviation (11.4 times) from commercial practice. This is not a criticism; the well-known requirements to satisfy Military Standards will often cost an order of magnitude more than a commercial performance equivalent.

It should be noted that the cost-predictor basis is inflation adjusted inasmuch as the cost forecasts are based on the actual cost of that year. It follows, then, that the predictor is inflation adjusted to the trend. The general trend that is obvious is that an order of magnitude improvement is seen in the cost vs speed relationship every ten years.

ACKNOWLEDGMENT

It should be obvious that no one individual could have expertise in so vast a subject as represented by this document. Therefore, we acknowledge the contributions of the following:

Dr. Raivo Vest for hardware architecture
Mr. William E. Burr for software architecture
Mr. James H. Edwards for technology
Dr. Martin Nisenoff for advanced technology
The CFA/MCF family for software architecture, evaluation, and analysis.

REFERENCES

1. Applied Technology, "Countermeasures Warning and Control System," Vol 2., System B, Applied Technology Div. (ITEK Corp.), Sunnyvale, Calif., Oct. 14, 1974.
2. Fawcett, D. G., "The Adaptive High Speed Signal Sorter," Final Report for ONR/NRL, FRN00014-75-C-1201, Nov. 1, 1977, Hughes Aircraft Company, Fullerton, CA.
3. Litton Industries (AMECON Div.) proposal, "ESM IFM Receiver and Processor for EP-3E," Litton (AMECON Div.), College Park, Md., July 1977.
4. Rockwell International, "Adaptive Processor for Electronic Reconnaissance," Rockwell International, Anaheim, Calif., June 1975.
5. Texas Instruments, "The Microprocessor Handbook," Texas Instrument, Dallas, Texas, 1975.
6. Osborne, Adam and Associates, "*An Introduction to Microcomputers,*" Vol. 1 *Basic Concepts*, Berkeley, Calif., 1976.
7. Wagner, James, E. Lieblein, J. Rodriguez, and H. S. Stone, "Evaluation of the Software Bases of the Candidate Architectures for the Military Computer Family," interim report issued by NRL Code 7590 on August 1, 1976, contained in *Final Report of the Army/Navy Computer Family Architecture Selection Committee*, as Chap. VI, Sec. II.
8. Burr, W. E., A. H. Coleman, and W. R. Smith, *Summary of the Final Report of the Army/Navy Computer Family Architecture Selection Committee,* NRL Code 7590 Dec. 1, 1976.
9. Motorola, "Introduction to Microprocessors," 1975.
10. Parasuraman, B., "High-Performance Microprocessor Architectures" *Proc. IEEE* **64**, No. 6, 851-859 (June 1976).
11. Advanced Micro Devices Corp., "The AM 2900 Family Data Book," Sunnyvale, Calif., 1976.
12. Advanced Micro Devices Corp., Mick, J. R., and Barr, J., "Microprogramming Handbook," Sunnyvale, Calif., 1976.
13. "ATAC-16N Principles of Operation," Applied Technology, Inc., II, Sunnyvale, Calif., Dec. 1976.
14. "Microprogramming Software for Hewlett-Packard 2100 Computer," Hewlett-Packard, Cupertino, Calif., Aug. 1972.
15. "AN/UYK-20 Technical Description," Sperry-Univac, PX10431C. St. Paul, Minn., Nov. 1976.
16. "AN/AYK-14 Technical Description," Control Data Corp., Minneapolis, Minn., 1977.

17. "AN/UYK-15 Technical Description," Sperry-Univac, PX7917A, St. Paul, Minn., Mar. 1973.
18. Fuller S. H., H. Stone, and W. E. Burr, "Selection of Candidate Architectures and Initial Screening." Vol. II of *Computer Family Architecture Selection Committee Final Report*, Naval Research Laboratory, Washington, D. C. 20375. Dec. 1, 1976.
19. Fuller, S. H., W. E. Burr, P. Shaman, And D. A. Lamb, "Evaluation of Computer Architectures via Test Programs," *AFIPS Conference Proceedings.*, **46**, 1977, National Computer Conference, Dallas, Texas.
20. Wagner, J., B., Lieblein, J. Rodriguez, H. S. Stone, "Evaluation of the Candidate Architectures for the Military Computer Family," *AFIPS Conference Proceedings*, **46**, 1977 National Computer Conference, Dallas, Texas.
21. Fuller, S. F., W. E. Burr, P. Shaman, D. Lamb, "Evaluation of Computer Architectures via Test Programs," Vol. III of *Computer Family Architecture Selection Committee Final Report*, NRL Code 7590, Washington, D. C., Dec. 1, 1976.
22. Popek, G. J., and R. P. Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures," *Commun. ACM*, **17**, No. 7, 412-421, July 1974.
23. "AYA-6 Computer Instruction Set," IBM 67-538029, Sept. 13, 1967, Federal Systems Div., Oswego, N. Y.
24. S. H. Fuller, G. Mathew, L. Szewerenko, "Comparative Evaluation of the MCF Computer Architectures," Dept. of Computer Sciences, Carnegie-Mellon Univ., Pittsburgh, Pa., Jan. 15, 1978.
25. Stone, H. S., "An Inventory of the Existing Software Base for Computer Architectures Considered in the MCF Project," University of Massachusetts, Electrical Engineering Dept. Feb. 23, 1977.
26. Smith, W. R., J. J. Cornyn, A. H. Coleman, W. Svirsky, R. Estell, and P. Sabin, "Life Cycle Cost Models for Comparing Computer Family Architectures," *AFIPS Conference Proceedings*, **46**, 1977 National Computer Conference, Dallas, Texas.
27. Cornyn, J. J., "Top-Down Life-Cycle Cost-Analysis Model for Selecting a Computer Family Architecture," NRL Code 7590, Washington, D. C., Aug. 1976.
28. "Submicron IC Technology: From Lab Curiosity to Production," *EDN* **22** (No. 9), 21-24 (May 5, 1977).
29. Faggin, F., "How VLSI Impacts Computer Architecture," *IEEE Spectrum*, May 1978.
30. Nisenoff, N., "The Engineer in the Information Environment of 2000 A. D.," Forecasting International, LTD Paper, *EASCON Proceedings*, Arlington, Va., Sept. 1976.

BIBLIOGRAPHY

ESM Systems

1. IBM, "TASES ELINT PROPOSAL"
2. Long, R. E., "Advanced EW Processing," AOC Technical Symposium, 1976.
3. Bigas, W. R., "Automated IFM/DF Handles Dense Threat Environment," *Electronic Warfare*, Mar. Apr. 1977, pp. 50-56.
4. Hyatt, G., "Signal Processing in the Frequency Domain," *Electronic Warfare*, **9** (No.5), Sept/Oct 1977, pp. 95-98.
5. Gunn, T. L., "Signal Sorting in Dense Environments," *Electronic Warfare*, **9** (No. 5) Sept/Oct 1977, pp. 95-96, 98.

Computers, Microprocessors

6. Frisch, I., et al., "7 Steps to Picking the Best Communications Processer," *Data Communications* Nov/Dec 1976, pp. 25-37.
7. Wolin, L., "Procedure Evaluates Computers for Scientific Applications," *Computer Design,* **15** (No. 11), 93-100 (Nov. 1976).
8. Ollivier, R. T., "A Technique for Selecting Small Computers," *Datamation* **16** 141-145 (Jan. 1970).
9. Osborne, Adam and Associates, *An Introduction to Microcomputers* Vol. 2, *Some Real Products,* Berkeley, Calif., 1976.
10. Ogdin, C. A., "μC Design Course" *EDN,* Nov. 20, 1976, pp. 127-136.
11. Bursky, D., "Microprocessor Selection Guide" *Electronic Design* **25**, Oct 11, 1977, pp. 55-67.
12. Gellender, E. "Learn Microprocessor Fundamentals," *Electronic Design* **25** (No. 21) Oct. 11, 1977, pp. 74-79.
13. Holland, S. "Break the 65-Kbyte Address Barrier," *Electronic Design* **25** (No. 23), Nov. 8, 1977, pp. 82-85.
14. Weissberger, A. J. "Analysis of Muitiple-Microprocessor System Architectures," *Computer Design,* June 1977, pp. 151-163.
15. Burton, T., "Multi-μP Systems Combine the Efficiency ----," *Electronic Design* **25** (No. 16) Aug. 2, 1977, pp. 68-71.
16. Gabriele, T., "Multiprocessing can Marry a Radar ----," *Electronic Design* **25** (No. 16), Aug. 2, 1977, pp. 74-77.
17. Balph, T., "Get the Best Processor Performance by Building it from ECL Bit Slices," *Electronic Design* 25 (No. 12), June 7, 1977, pp. 84-92.
18. Clymer, J., "Use 4-bit Slices to Design Powerful Microprogrammed Processors," *Electronic Design* **25** (No. 10), May 10, 1977, pp. 62-71.
19. Nemec, J., "A Primer on Bit-Slice Processors" *Electronic Design* **25** (No. 3), Feb. 1, 1977, pp. 52-60.
20. Nemec, J. and Lav, S.Y., "Bipolar Microprocessors — An Introduction to Architecture and Applications," *EDN* **22**(No. 19), Oct. 5, 1977, pp. 79-81; **22**(No. 17), Sept. 20, 1977, pp. 63-67.
21. Bass, J. E., "A Peripheral-Oriented Microcomputer System," *Proc. IEEE* **64**(No. 6), pp. 860-873, June 1976.
22. Nichols, A. J., "An Overview of Microprocessor Applications," *Proc. IEEE* **64** (No. 6), 951-953, June 1976.
23. Lin, W. C., "Microprocessor-Based Digital System Design Fundamentals and the Development Laboratory for Hardware Designers and Engineering Executives," *Proc. IEEE* **65**(No. 8), 1138-1161, Aug. 1977.

Computers, Manufacturers' Literature

24. Intel, "8080 Microcomputer Systems Manual," Jan. 1975.
25. Hewlett-Packard, "A Pocket Guide to the Hewlett-Packard 2100A Computer."
26. "Bipolar Microcomputer Components Data Book," Texas Instruments, Jan. 1977.
27. Signetics, "Introducing the 3000."
28. Intel "Programming Manual for the 8080 Microcomputer System," May 1974.

Computers, E.W.

29. Allison, A. A., "Computers in Electronic Warfare," *Electronic Warfare*, Sept. 1976, pp. 31-34.
30. "World's First EW Minicomputer Debuts," *Electronic Warfare*, Sept. 1976, pp. 62-73.
31. "Make or Buy your Own Computer," *Electronic Warfare*, Jul/Aug 1974, pp. 27-35.
32. Jordan, G. B. "The Hardware Interface: Computer to Jammer," *Electronic Warfare*, Jul/Aug 1974, pp. 37-50.

Advanced Techniques

33. Jack, M. A., et al., "Waveform Detection and Classification with SAW Cepstrum Analysis," *IEEE Trans.* **AES-13**(No. 6), 610-619, Nov. 1977.
34. Gholson, N. H., and Moose, R. L., "Maneuvering Target Tracking Using Adaptive State Estimation," *IEEE Trans.* **AES-13**(No. 3), 310-317 (May 1977).
35. Friedland, B., "Optimum Steady-State Position and Velocity Estimation Using Noisy Sampled Position Data," *IEEE Trans.* **AES-9**, 906-911 (Nov. 1973).
36. Foy, W. H., "Position-Location Solutions by Taylor-Series Estimation," *IEEE Trans.* **AES-12**(No. 2), 187-193 (Mar. 1976).
37. Singer, R. A., and Behnke, K. W., "Real-Time Tracking Filter Evaluation and Selection for Tactical Applications," *IEEE Trans.* **AES-7**(No. 1), 100-110 (Jan. 1971).
38. Pearson, J. B., III, and Stear, E. B., "Kalman Filter Applications in Airborne Radar Tracking," *IEEE Trans.* **AES-10,** 319-329 (May 1974).
39. Widrow, B., et al., "Adaptive Noise Cancelling: Principles and Applications," *Proc. IEEE* **63**(No. 12), 1692-1716 (1975).
40. "Comparison of the AN/UYK-20(V) and AN/UYK-28 Computers for ROC Applications," Sperry-Univac, Mar. 1977.
41. "Comparison of the AN/UYK-20 and AN/UYK-19 Computers for ROC Applications," Sperry-Univac.
42. Grumman, "EXCAP Memory Usage," private correspondence.
43. Shimp, A. C., "Can Core Survive," *Digital Design*, Nov 1977, pp. 31-38.

Software Architecture

44. Amdahl G. M., G. A. Blaauw, and F. P. Brooks, Jr., "Architecture of the IBM System/360," *IBMJ. R and D***8**, April 1964, pp. 87-101.
45. *Computer Review*, formerly *Computer Characteristics Review*, GML Corporation, Lexington, Mass., 02173, 1975.
46. Stone, H. S., "An Audit of the Selection Criteria for Computer Family Architecture," CFA Memorandum, January, 1976. Distrubuted at the 18-20 February CFA meeting.
47. Fuller, S. F., H. S. Stone, and W. E. Burr, "Initial Selection and Screening of the CFA Candidate Computer Architecture," *AFIPS Conference Proceedings*, **46**, 1977 National Computer Conference.
48. Lucas, H. C., "Performance Evaluation and Monitoring," *ACM Computing Surveys* **3** (No. 3), 1971, pp. 79-91.
49. Bernwell, N. (ed.), *Benchmarking: Computer Evaluation and Measurement*, John Wiley & Sons, New York, 1975.
50. Wichmann, B. A., *Algol 60 Compilation and Assessment*, Anderson Press, New York, 1973.

51. Bell, C. G., and A. Newell, compilers, *Computer Structures: Readings and Examples*, McGraw-Hill, New York, 1971.
52. *Computer Review*, GML Corporation, Lexington, Mass., 1976.
53. Stone, H. S. (ed.), *Introduction to Computer Architecture*, Science Research Associates, Chicago, 1975.
54. Davies, O. L. (ed.), *The Design and Analysis of Industrial Experiments*, Oliver and Boyd, Edinburgh, 1971.
55. Anderson, V. L., and R. A. McLean, *Design of Experiments, a Realistic Approach*, Marcel Dekkerk, Inc., New York, 1974.
56. Connor, W. S., and M. Zelen, *Fractional Factorial Experiment Designs for Factors at Three Levels*, National Bureau of Standards, Applied Mathematics Series 54, 1959.
57. Rao, C. R., *Linear Statistical Inference and its Applications*, 2nd ed., John Wiley & Sons, New York, 1973.
58. Cornyn, J. J., W. R. Smith, W. R. Svirsky, and A. H. Coleman, "Two Life-Cycle Cost Models for Comparing Computer Architectures," *AFIPS Conference Proceedings*, **46**, 1977 National Computer Conference, Dallas, Texas.
59. Box, G. E. P. and D. R. Cox, "An Analysis of Transformations," *J. Roy. Statist. Soc.* Series B., **26**, 1964, pp. 211-252.
60. Barbacci, M. R., "The Symbolic Manipulation of Computer Descriptions: ISPL Compiler and Simulator," Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., 1976.
61. W. S. Connor and Shirley Young, *Fractional Factorial Designs for Experiments with Factors at Two and Three Levels*, National Bureau of Standards, Applied Math Series 58, Sept. 1, 1961.
62. "Military Computer Family, Selection Methods for a Computer Family Architecture," reprinted from *AFIPS Conference Proceedings* **46** AFIPS Press, Montvale, N. J., 1977.
63. H. Scheffe, *The Analysis of Variance*, John Wiley and Sons, Inc., New York, 1959.
64. Salisbury, A. B., "MCF: A Military Computer Family for Computer-Based Systems," *Signal* **30**(No. 9), 42-45 (July 1976.)
65. Coleman, A. H., "Army/Navy Military Computer Family," *Digest of Papers of COMPCON 76.* IEEE. Cat. No. 76 CH1115-5C, Thirteenth IEEE Computer Society International Conference, Washington, D. C., Sept. 7-10, 1976, pp. 230-232.
66. Estell, R. G., R. P. Sabin, and W. R. Smith, "Final CFA Selection Methodology Subcommittee Preliminary Report," April 1976.
67. Cornyn, J. J., W. R. Smith, A. H. Coleman and W. Svirsky: "Life Cycle Cost Models for Comparing Computer Family Architectures," *AFIPS Conference Proceedings* **46**, 1977 National Computer Conference, Dallas, Texas.
68. Svirsky, W., T. Giles, and A. Irwin, "Life Cycle Cost Analysis of Computer Family Architecture (CFA) Finalists Within Army Embedded Computer Systems," System Development Corporation, unpublished manuscript generated for CFA Selection Committee, Aug. 1976.
69. SADPR-85 Study Group, "Support of Air Force Automatic Data Processing Requirements through the 1980's (SADPR-85)," Electronics Systems Division, Hq. ESD (MCS), Hanscom AFB, Mass. Six Volumes, ESD-TR-74-192. Vols. 1 and 3 dated Jun 1974.
70. R. Turn, *Computers in the 1980's*, Columbia Univ. Press, New York and London, 1974.
71. Fuller, S. H., W. E. Burr, P. Shaman, and D. A. Lamb, "Evaluation of Computer Architectures via Test Programs," *AFIPS Conference Proceedings* **46**, 1977 National Computer Conference, Dallas, Texas.

72. Fisher, D.A., "Automatic Data Processing Costs in the Defense Department," Institute for Defense Analyses, Arlington, Va., IDA Paper P-1046, Oct. 1974.
73. Wagner, J., et al., "Procedure for the Results of the Evaluation of the Software Bases of the Candidate Architectures for the Military Computer Family," 6 August 1976, prepared by the Software Evaluation Subcommittee, NRL.
74. Shishko, R., "Choosing the Discount Rate for Defense Decision Making," Rand Corp., Santa Monica, Calif. R-1953-RC, July 1976.
75. AFR-172-2; DODI 7041.3, "Economic Analysis of Proposed Investments," 30 December 1969, Attachment 2, p. 42.
76. Kossiakoff, A., T.P. Sleight, E.C. Prettyman, J.M. Park, and P.L. Hazan, "DOD Weapon Systems Software Management Study," Johns Hopkins Univ., Applied Physics Lab, Laurel, Md., June 1975, APL/JHU-SR 75-3, AD-AO22 160/6WC. Abstract in Comp., Control, and Info. Theory, May 3, 1975.
77. Chapin, G.G., "What is Different About Tactical Military Operational Programs," *AFIPS Conf. Proc.* **42**, 1973, Nat. Comp. Conf. pp. 787-795.
78. Premo. A.F., Jr., "Computer Software: Estimating Guidelines," *COMPCON 76, Digest of Papers*, IEEE Pub. No. 76CH1115-5C, Sept. 7-10, 1976, pp. 146-151.
79. Boehm, B.W., "Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980's (CCIP-85). Vol. IV, Technology Trends: Software," Space and Missile Systems Organization, AFSC, Los Angeles, Calif., Oct. 1973, AD919267L.
80. Boehm, B.W., et al., "Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980's (CCIP-85). Vol. I, Highlights," April 1972, SAMSO/XRS 71-1. U.S. Air Force, AD-900031L.
81. Private communication, meeting of D. Fisher, W. Smith, and J. Cornyn on May 3, 1976.
82. McLaughlin, R.A., "1976 DP Budgets," *Datamation* **22** (No.2), February 1976, pp. 52-58.
83. System Development Corp., "Embedded Computer System Data Processing Requirement for U.S. Army Weapon/Data Systems," Draft, Mar. 1 1976, prepared for CENTACS, U.S. Army Electronics Command, Ft. Monmouth, N.J. Contract DAAB07-76-C-0334.
84. Grosch, H.R.J., "High Speed Arithmetic: The Digital Computer as a Research Tool," *J. Optical Soc. Amer.* **43** Apr. 1953, pp. 306-310.
85. Withington, F.G., "Beyond 1984 — A Technology Forcast," *Datamation*, Jan. 1975, pp. 54-73.
86. Private communication with Al Irwin, W. Svirsky, and T. Giles of System Development Corporation.
87. Young, H.D., *Statistical Treatment of Experimental Data*, McGraw-Hill Book Company, Inc., New York, 1962.
88. Boehm, B.W., "Keytone Address: The High Cost of Software," TRW, Redondo Beach, Calif., in *Proceedings of a Symposium on the High Cost of Software*, Sept. 17-19, 1973, at Naval Postgraduate School, Monterey, Calif., Stanford Research Institute, Menlo Park, Calif., SRI Proj. 3272, pp. 27-40.
89. Brooks, F.P., Jr., *The Mythical Man-Month*, Addison-Wesley Publishing Co., Reading Mass., 1975.
90. Manley, J.H., "Embedded Computers—Software Cost Considerations," *AFIPS Conf. Proc.* **43**, 1974, pp. 343-347.
91. Taback, M.A., and M.C. Ditmore, "Estimation of Computer Requirements and Software Development Costs," General Research Corp., Santa Barbara, Calif. RM-1873, March 1974, AD-782-2208WC, p. 20.

92. Wolverton, R.W., "The Cost of Developing Large-Scale Software," *IEEE Trans.* **C-23** (No. 6), pp. 615-636, June 1974.
93. Smith, W.R., "AADC Computer Family Architecture News **4** (No. 3), Sept. 1975, pp. 15-21.

Microelectronics

94. J.L. Hilburn and P. N. Julich. "Microcomputers/Microprocessors: Hardware, Software, and Applications," Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976.
95. "Special Issue on Microprocessor Technology and Applications," *Proc. IEEE,* Vol. **64** (No. 6), (June 1976).

Microelectronic Circuit Elements

96. J.F. Gibbons, *Semiconductor Electronics,* McGraw-Hill Book Company, New York, 1966.
97. A.S. Grove, *Physics and Technology of Semiconductor Devices,* John Wiley & Sons, Inc., New York, 1967.
98. Sorab K. Ghandhi, *The Theory and Practice of Microelectronics,* John Wiley & Sons, Inc., New York, 1968.
99. J.D. Meindl, *Micropower Circuits,* John Wiley & Sons, Inc., 1969.
100. D. Hamilton and W. Howard, *Basic Integrated Circuit Engineering,* McGraw-Hill Book Company, 1975.

Large-Scale Integration of Microelectronic Circuits

101. R.G. Hibberd, *Integrated Circuits: A Basic Course,* McGraw-Hill Book Company, 1969.
102. W.C. Hittinger, "Metal-Oxide-Semiconductor Technology," in *Scientific American* **229** (No. 2), 48-57 (Aug. 1973).
103. H.W. Gschwind and E.J. McCluskey, *Design of Digital Computers,* Springer-Verlag, 1975.
104. A.G. Vacroux, "Microcomputers," in *Scientific American* **232** (No. 5), 32-40 (May, 1975).
105. S. Middelhoek and P. Dekker, *Physics of Computer Memory Devices,* Academic Press, Inc., 1976.
106. E.W. McWhorter, "The Small Electronic Calculator," in *Scientific American* **234** (No. 3), 88-96,98 (Mar. 1976).

Fabrication of Microelectronic Circuits

107. A.B. Grebene, "Integrated Circuit Fabrication Processes," in *Analog Integrated Circuit Design.* Van Nostrand Reinhold Company, New York, 1972.
108. "MOS Integrated Circuits: Theory, Fabrication, Design, and Systems Applications of MOS LSI," Engineering staff of American Micro-systems, Inc., edited by W.M. Penny and L.Lau. Van Nostrand Reinhold Company, New York, 1972.
109. "MOS/LSI Design and Application," W.N. Carr and J.P. Mize, R.E. Sawyer and J.R. Miller, eds. McGraw-Hill Book Company, New York, 1972.
110. W.C. Hittinger, "Metal-Oxide-Semiconductor Technology," in *Scientific American* **229** (No. 2) 48-57 (Aug. 1973).

Microelectronic Memories

111. *Semiconductor Memories,* D.A. Hodges, ed. IEEE Press, New York, 1972.
112. Rein Turn, "Mass Memories," in *Computers in the 1980s.* Columbia University Press, New York, 1974.
113. Rein Turn, "Random-Access Memories," in *Computers in the 1980s.* Columbia University Press, New York, 1974.
114. C.H. Sequin and M.F. Tompsett, "Digital Memories," in *Charge Transfer Devices,* Academic Press, Inc., New York, 1975.
115. *Magnetic Bubble Technology: Integrated-Circuit Magnetics for Digital Storage and Processing.* Hsu Chang, ed. IEEE Press, New York, 1975.
116. "Special Issue on Large Capacity Digital Storage Systems," *Proc. IEEE* **63** (No. 8) Aug. 1975.
117. J.A. Rajchman, "New Memory Technologies," in *Science,* **195** (No. 4283), 1223-1229 (Mar. 18, 1977).

Microprocessors

118. "Special Issue on Microprocessor Technology and Applications," *Proc. IEEE,* **64** (No. 6) (June 1976).
119. "Special Issue on Small Scale Computing." *Computer* **10**, (No. 3) (Mar. 1977).
120. S.E. Madnick, "Trends in Computers and Computing: The Information Utility," in *Science* **195** (No. 4283), 1191-1199 (Mar. 18, 1977).

The Role of Microelectronics in Data Processing

121. T.C. Bartee, *Digital Computer Fundamentals,* 3rd ed., McGraw-Hill Book Company, New York, 1972.
122. C. Weitzman, *Minicomputer Systems: Structure, Implementation and Application,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1974.
123. T.A. Dolotta, M.I. Bernstein, R.S. Dickson, Jr., N.A. France, B.A. Rosenblatt, D.M. Smith and T.B. Steel, Jr., *Data Processing in 1980-1985: A Study of Potential Limitations to Progress,* John Wiley & Sons, Inc., New York, 1976.
124. D.A. Hodges, "Trends in Computer Hardware Technology," in *Computer Design* **15** (No. 2), 77-85 (Feb. 1976).

The Role of Microelectronics in Instrumentation and Control

125. B.M. Oliver, "Servo-Motor Response," in *Proc. IEEE* **53** (No. 2), 201-202 (Feb. 1965).
126. *Electronic Measurements and Instrumentation,* M. Oliver and J.H. Cage, eds., McGraw-Hill Book Company, 1971.
127. W. Banks and J. C.. Majithia, "Microprocessors: Design and Applications in Digital Instrumentation and Control," in *IEEE Trans.* **IM-25** (No. 3), 245-249, Sept. 1976.
128. A. Santoni, "Digital Systems Spawn New Tasks in Measurement," in *Electronics* **49** (No. 22), 100-106 (Oct. 28, 1976).
129. "LSI Chips Taking Over More Household Chores," G.M. Walker in *Electronics* **49** (No. 22), 128-134 (Oct. 28 1976).

The Role of Microelectronics in Communication

130. *Principles of Pulse Code Modulation,* K.W. Cattermole, American Elsevier Publishing Company, Inc., New York, 1969.
131. N.S. Jayant, "Digital Coding of Speech Waveforms: PCM, DPCM, and DM Quantizers," in *Proc. IEEE* **62** (No. 5), 611-632 (May 1974).
132. I. Jacobs and S.E. Miller, "Optical Transmission of Voice and Data," in *IEEE Spectrum* **14** (No. 2), 32-41 (Feb. 1977).
133. "Special Issue: The 1A Processor," *Bell Sys. Tech. J.* **56** (No. 2) (Feb. 1977).
134. W. Peil and R.J. McFadyen, "Single-Slice Superhet," in *IEEE Spectrum* **14** (No. 3), 54-57 (Mar. 1977).

Microelectronics and Computer Science

135. E.F. Moore, "The Shortest Path Through a Maze," *The Annals of the Computation Laboratory of Harvard University: Vol. XXX, Proceedings of an International Symposium on the Theory of Switching, Part II.* Harvard University Press, Cambridge, Mass. 1959.
136. B. Hoeneisen and C.A. Mead, "Fundamental Limitations in Microelectronics, I: MOS Technology," in *Solid-State Electronics* **15** (No. 7), 819-829 (July 1972).
137. B. Hoeneisen and C.A. Mead, "Limitations in Microelectronics, II: Bipolar Technology," in *Solid-State Electronics* **15** (No. 8), 891-897 (Aug. 1972).
138. I. E. Sutherland D. Oestreicher, and "How Big Should a Printed Circuit Board Be?" in *IEEE Trans.* **C-22** (No. 5), 537-542 (May 1973).

Microelectronics and the Personal Computer

139. J. S. Bruner, "Towards a Theory of Instruction," Belknap Press of Harvard University Press, Cambridge, Mass., 1966.
140. S. A. Papert and M. Minsky, "Artificial Intelligence," Condon Lectures, Oregon State System of Higher Education, 1974.
141. A. Kay and A. Goldberg, in "Personal Dynamic Media," **10** (No. 3), 31-41 (Mar. 1977).
142. Torrero, E. A., "Solid-State Devices," *IEEE Spectrum*, (Jan. 1977).

Appendix A

DERIVATION OF TABLE IV-9

Table IV-9 represents the number of thousands of times each second that a given configuration of processor type and memory type can perform the simple Benchmark Flywheel tracking algorithm presented in Section II.C.2.

The equations to be solved are

$$\hat{S}_p(K+1) = \hat{S}_{pk} + \frac{\hat{S}_{mk} - \hat{S}_{pk}}{16} \tag{A1}$$

$$\hat{D}(k+1) = \hat{D}_k + \frac{|\hat{S}_{mk} - \hat{S}_{pk}| - \hat{D}_k}{16} \tag{A2}$$

$$3\hat{\sigma}(k+1) \approx 3\text{–}3/4\ \hat{D}(k+1) = 4\ \hat{D}(k+1) = 1/4\hat{D}(k+1) \tag{A3}$$

$$\hat{L.L.} = \hat{S}_p(k+1) - 3\hat{\sigma}(k+1) \tag{A4}$$

$$\hat{U.L.} = \hat{S}_p(k+1) + 3\hat{\sigma}(k+1) \tag{A5}$$

where

$\hat{S}_{pk}$ is the kth estimate of the signal parameter vector, consisting of three signal parameters.

$\hat{S}_{mk}$ is the kth measured value of the signal parameter vector.

$\hat{D}_k$ is the mean absolute deviation of the signal parameter vector from the estimated value.

$3\hat{\sigma}(k+1)$ is an approximation to the 3σ interval generally used to set the acceptance windows. 3σ represents three standard deviations in a normally distributed variable.

$\hat{L.L.}$ is the lower limit of the acceptance window.

$\hat{U.L.}$ is the upper limit of the acceptance window.

The three signal parameters chosen for the example are TOA, AOA, and frequency, although any other three parameters may be used to perform the signal sorting and tracking function.

In the case of TOA, which is continually increasing, it is not practical to keep a running average. Therefore, Eq. (A1) must be modified to calculate the running average of the TOA differences — or the PRI. Therefore, for the TOA calculation, Eq. (A1) must be modified as follows:

$$PRI(k+1) = PRI(k) + \frac{(TOA)mk - (TOA)pk}{16}$$

$$(TOA)p(k+1) = (TOA)pk + PRI(k+1).$$

A. Software Solutions

1. Technique

The program steps required to perform these calculations are given in Table A1. The table was constructed under the following assumptions:

- The microprocessor contains two independent accumulators, A and B, each capable of performing any microprocessor function.

- The data memory addressing scheme is assumed to be absolute addressing, so that the data address register does not have to be updated or incremented. This could be accomplished by data paging, where the page number is the emitter number being processed and is determined by hardware or software external to the processor.

- All instructions require a single instruction word. In the load and store instructions, the address (within the page) is part of the instruction word.

- All data are stored in integer format, so that multiplication and division by powers of 2 can be accomplished by shifting left or right, respectively.

Program steps 10 and 11 in Table A1 are required only when the TOA calculation is made, and are to be skipped for the other calculations. Therefore the normal parameter update calculations require 32 program steps, 10 of which reference the memory. The TOA update calculations require 34 program steps, 12 of which reference the memory.

If all parameters are to be updated in parallel, the program requires 34 program steps, 12 of which reference the memory. The non-TOA parameter calculations remain idle (or address dummy memory locations) during steps 10 and 11. If the three parameters are to be updated in series, steps 10 and 11 are skipped for the non-TOA calculations and the program requires 98 program steps, 32 of which reference the memory.

For the simple move, add, and shift instructions in Table A1, the time required to execute one computer instruction in the register-to-register mode is given by (see Section VI.A)

$$T_c = T_m + T_p \tag{A6}$$

where T_m is the time required to fetch the instruction, and T_p is the time required to execute the instruction.

For the memory reference instruction, an additional memory cycle time is required to fetch the data, so if the data and instruction read/write times are the same,

$$T_c = 2T_m + T_p. \tag{A7}$$

For the Benchmark Program, the data address (within the page) location is assumed to be included within the instruction word, so that an additional address fetch cycle is not required.

Adding up the time required to perform the instructions in Table A1, using Eqs. (A6) and (A7) (assuming T_m and T_p are the same for each instruction used) yields the total time required to run the program:

Table A1 — Steps Required to Perform the Benchmark Program

Step	Fuction	Operation	Register A	Contents B
1.*	Load A with $\hat{S}_{mk}$		$\hat{S}_{mk}$	—
2.*	Subtract $\hat{S}_{pk}$ from A		$\hat{S}_{mk}-\hat{S}_{pk}$	—
3.	Arith Shift Right A		$(\hat{S}_{mk}-\hat{S}_{pk})/2$	—
4.	Arith Shift Right A	Divide A by 16	$(\hat{S}_{mk}-\hat{S}_{pk})/4$	—
5.	Arith Shift Right A		$(\hat{S}_{mk}-\hat{S}_{pk})/8$	—
6.	Arith Shift Right A		$(\hat{S}_{mk}-\hat{S}_{pk})/16$	—
7.	Move A to B		$(\hat{S}_{mk}-\hat{S}_{pk})/16$	$(\hat{S}_{mk}-\hat{S}_{pk})/16$
8.	Skip next instruction if positive		$(\hat{S}_{mk}-\hat{S}_{pk})/16$	$(\hat{S}_{mk}-\hat{S}_{pk})/16$
9.	Two's complement of B		$(\hat{S}_{mk}-\hat{S}_{pk})/16$	$-(\hat{S}_{mk}-\hat{S}_{pk})/16$
10.*	Add PRI to A	Skip if not TOA	PRI$(k+1)$	$\lvert\hat{S}_{mk}-\hat{S}_{pk}\rvert/16$
			PRI$(k+1)$	$\lvert\hat{S}_{mk}-\hat{S}_{pk}\rvert/16$
11.*	Store A in (PRI)		PRI$(k+1)$	$\lvert\hat{S}_{mk}-\hat{S}_{pk}\rvert/16$
12.*	Add $\hat{S}_{pk}$ to A		$\hat{S}_p(k+1)$	$\lvert\hat{S}_{mk}-\hat{S}_{pk}\rvert/16$
13.*	Store A in $\hat{S}_{pk}$		$\hat{S}_p(k+1)$	$\lvert\hat{S}_{mk}-\hat{S}_{pk}\rvert/16$
14.*	Load A with $\hat{D}_k$		$\hat{D}_k$	$\lvert\hat{S}_{mk}-\hat{S}_{pk}\rvert/16$
15.*	Add A to B		$\hat{D}_k$	$\hat{D}_k+\lvert\hat{S}_{mk}-\hat{S}_{pk}\rvert/16$
16.	Arith. Shift Right A		$\hat{D}_k/2$	$\hat{D}_k+\lvert\hat{S}_{mk}-\hat{S}_{pk}\rvert/16$
17.	Arith. Shift Right A	Divide by 16	$\hat{D}_k/4$	$\hat{D}_k+\lvert\hat{S}_{mk}-\hat{S}_{pk}\rvert/16$
18.	Arith. Shift Right A		$\hat{D}_k/8$	$\hat{D}_k+\lvert\hat{S}_{mk}-\hat{S}_{pk}\rvert/16$
19.	Arith. Shift Right A		$\hat{D}_k/16$	$\hat{D}_k+\lvert\hat{S}_{mk}-\hat{S}_{pk}\rvert/16$
20.	Subtract A from B		$\hat{D}_k/16$	$\hat{D}(k+1)$
21.*	Store B in $\hat{D}_k$		$\hat{D}_k/16$	$\hat{D}(k+1)$
22.	Move B to A		$\hat{D}(k+1)$	$\hat{D}(k+1)$
23.	Arith. Shift Left A	Multiply A by 4	$2\,\hat{D}(k+1)$	$\hat{D}(k+1)$
24.	Arith. Shift Left A		$4\,\hat{D}(k+1)$	$\hat{D}(k+1)$
25.	Arith. Shift Right B	Divide B by 4	$4\,\hat{D}(k+1)$	$\hat{D}(k+1)/2$
26.	Arith. Shift Right B		$4\,\hat{D}(k+1)$	$\hat{D}(k+1))/4$
27.	Subtract B from A		$3\,\sigma(k+1)$	$\hat{D}(k+1)/4$
28.*	Store A in (3σ)		$3\sigma(k+1)$	$\hat{D}(k+1)/4$
29.*	Load B from $(\hat{S}_{pk})$		$3\sigma(k+1)$	$\hat{S}_p(k+1)$
30.	Subtract A from B		$3\sigma(k+1)$	L.L.
31.*	Store B in (L.L.)		$3\sigma(k+1)$	L.L.
32.	Add A to B		$3\sigma(k+1)$	$\hat{S}_p(k+1)$
33.	Add A to B		$3\sigma(k+1)$	U.L.
34.*	Store B in (U.L.)		$3\sigma(k+1)$	U.L.

Notes: $(\hat{S}_{mk})$ refers to the contents of the memory location that holds the data value for $\hat{S}_{mk}$).
* represents external memory access.

Serial Program

$$T_{ts} = 130T_m + 98T_p \tag{A8}$$

Parallel Program

$$T_{tp} = 46\ T_m + 34\ T_p. \tag{A9}$$

For a microprogrammed computer, where the entire sequence is microprogrammed, the individual instructions need not be fetched, so there is a saving of one T_m for each instruction after the first (the first instruction must be fetched to start the sequence). The resulting time required to run the program is:

Serial Microprogrammed Computer

$$T_{ms} + 33\ T_m + 98\ T_p \tag{A10}$$

Parallel Programmed Program

$$T_{mp} = 13\ T_m + 34\ T_p. \tag{A11}$$

The total times calculated are the times required to completely process one pulse by the Benchmark Program. The number of emitters processed, assuming 1000 pulses per emitter, is

$$N_e = Int\left(\frac{1}{1000T}\right) \tag{A12}$$

where $Int(x)$ is the integer value of x.

2. The Processors

a. NMOS processor

The computer chosen for the NMOS processing example is the CP 1600 microprocessor made by General Instruments coupled with an MM 2102-type static NMOS memory.

The memory has a read/write cycle time of 1.0 μs maximum. The processor takes four internal clock periods to read the memory (to fetch the instruction), and about two additional clock periods to execute simple instructions. There are two versions of the processor: the 1600 with a minimum clock period of 300 ns, and the 1600A with a minimum clock period of 200 ns. Assuming a "halfway" clock period of 250 ns for the processor yields

$$T_m = 2.0\ \mu s$$
$$T_p = 1.0\ \mu s. \tag{A13}$$

Note that, since the microprocessor allots 2 μs to read the memory, the effective memory cycle time is 2 μs, even though the actual memory cycle time is 1 μs. Substituting Eq. (A13) into (A8) through (A12) yields the values presented in Table A2.

Table A2 — Signal Tracking Capability of Various Processors Number

Processor	Memory	Number of Emitters			
		General Purpose		Microprogrammed	
		Series	Parallel	Series	Parallel
SOFTWARE					
NMOS	NMOS	2	7	6	16
Bipolar	Core	6	18	17	46
AN/UYK-20	Core	10			
Bipolar	Fast NMOS	11	31	24	66
Bipolar	Bipolar	17	50	30	85
		Number of Emitters			
		Serial Memory		Parallel Memory	
Processer	Memory	Series	Parallel	Series	Parallel
HARDWARE					
TTL	Core	36	94	133	400
TTL	Low-Power TTL	105	333	350	1052
TTL	TTL	196	512	455	1395

b. Bipolar processor

The processor selected for the bipolar processing example is the Advanced Micro Devices AM 2900 chip set. The processor data sheets specify a minimum clock period of 120 ns for the processor clock, based on delays within the processor itself. When the delays in the external circuitry that must be included to complete the computer are added to the processor's internal delays, a processor clock period of 200 to 300 ns is a more realistic value for reliable operation. The read/write cycle time for a bipolar RAM (using the 7489 as an example) is about 80 ns. The read/write cycle time for the low-power 74L89 is about 165 ns (max). Using these values as a guide, the cycle times for the all bipolar computer are assumed to be

$$T_m = 250 \text{ ns}$$
$$T_p = 250 \text{ ns}. \tag{A14}$$

c. Bipolar-core processor

A good high-speed, modern magnetic core array has an access time under 1 μs, which is about the same as a relatively slow MOS memory like the 2102. High-speed MOS memory read/write cycle times of 500 ns are now common. With this in mind, the following values may be assigned to the mixed bipolar-MOS processor:

Bipolar core or slow MOS

$$T_m = 1.0 \ \mu\text{s}$$
$$T_p = 250 \text{ ns} \tag{A15}$$

Bipolar — fast MOS

$$T_m = 500 \text{ ns}$$

$$T_p = 250 \text{ ns}. \tag{A16}$$

Substituting Eqs. (A13) through (A16) into Eqs. (A8) through (A12) yields the values presented in Table A2.

B. Hardware Solutions

The hardware solution to Eqs. (A1) through (A5) is presented in block diagram form in Fig. A1. The hardware solution parallels the software solution in detail, except that shift instructions are replaced by changing the connections between blocks and are not apparent in the diagram. Instruction 8, which takes the absolute value of Register B (Table A1) is replaced by an add/subtract circuit whose function is selected by the most significant bit of $(\hat{S}_{mk}-\hat{S}_{pk})$. As before, each memory is paged and stepped by external circuitry, which is not shown. The hardware solution can be broken down into steps just as the software solution was. These signal flow "steps" depend on whether the quantities to be processed are stored and accessed serially from the main memory, or whether they are available in parallel, each in its own memory module.

Table A3 shows the memory timing for the serially accessed memory case. The sequence consists of 4 memory read steps, 4 latch steps, 7 add/subtract steps, and 7 memory write steps. The memory write steps can run concurrently with the add/subtract steps, since the hardware computations need not be halted to perform the memory write cycles. Therefore, the total timing of the hardware circuit need only choose the largest of the following combinations, in addition to all the other steps listed:

Step 9 or Step 10

Step 13 or Step 14 and 15

Step 16 or Step 17

Step 18 or Step 19.

In the case of the serial computation, Steps 6 through 9 are deleted for the non-TOA calculations. If the same hardware is used for both the TOA and the non-TOA calculations, a data selector must be added to the circuit between Steps 9 and 10 to bypass calculations 6 through 9.

Since the memory is addressed serially (all blocks labeled "memory" in Fig. A1 are the same common memory module), additional time must be added for setting up the address of the next memory location in the address counter. Also, the memory module must be clocked to insure that there is sufficient delay between the initiation of consecutive memory read/write cycles for the data to stabilize.

The maximum throughput delay for the various blocks in Fig. A1 is shown in Table A4 (taken from the *National Semiconductor TTL Data Book*, 1976 Edition). To account for additional delays for wiring capacitance, circuit layout, etc., of a practical circuit application, 20 ns

Table A3 — Hardware Timing Diagram for Serially Accessed Memory

	Function	Clock Ticks	
		7489	74L89
1.*	Get $\hat{S}_m$	2	3
2.	Latch $\hat{S}_m$	1	1
3.*	Get $\hat{S}_p$	2	3
4.	Latch $\hat{S}_p$	1	1
5.	Subtract $(\hat{S}_m - \hat{S}_p)$	1	1
6.*	Get PRI	2	3
7.	Latch PRI	1	1
8.	Add (PRI)	1	1
9.*	Store PRI	2	3
10.	Add $(\hat{S}_p[k+1])$	—	—
11.*	Get $\hat{D}_k$	2	3
12.	Latch $\hat{D}_k$	1	1
13.*	Store $\hat{S}_p(k+1)$	2	3
14.	Add/Subtract $(\hat{D}_k + \lvert\hat{S}_m - \hat{S}_p\rvert/16)$	—	—
15.	Subtract $\hat{D}(k+1)$	—	—
16.*	Store $\hat{D}(k+1)$	2	3
17.	Subtract $(3\hat{\sigma})$	—	—
18.	Store $3\hat{\sigma}$	2	3
19.	Add (U.L) and Subtract (L.L)	—	—
20.*	Store U.L.	2	3
21.*	Store U.L.	2	3
	Total	26	36

(6.*–9.*: used for TOA only)

*Represents external memory access.

Table A4 — Maximum Throughput Delay for the Various Processing Blocks in Tables A3 and A5

Function	TTL Number	Max. Delay (ns)	Assumed Delay (ns)
Bipolar Memory	7489	80	150*
Bipolar Memory	74L89	165	250*
Core Memory	—	—	1000
Latch (J-K flip-flop)	7474	40	60
Add/Subtract	74181, 74182	50	70
Data Selector	74157	14	35
Counter	74191	36	55

*Includes address set-up time in the counter.

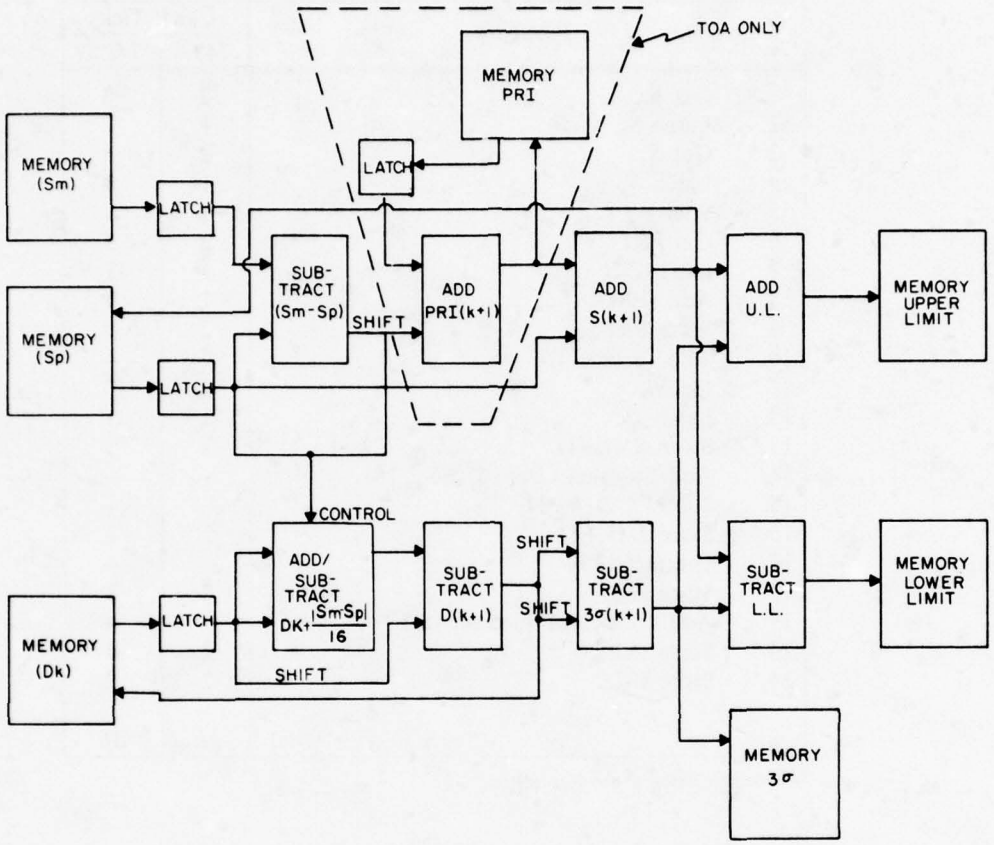


Fig. A-1—Hardware processor

are added to each published delay in the column labeled "assumed delay." The memory "assumed delay" times also include the assumed address set-up time of about 55 ns. Using these values in Table A4 in the timing diagram, and assuming clocks as follows:

75-ns clock for fast bipolar memory (7489)

83.3-ns clock for low-power memory (74L89)

100-ns clock for core memory

yields the data in Table A2.

The fastest possible way to process the Benchmark Program is to arrange the memory in such a manner that all data necessary to process one signal parameter can be accessed simultaneously. A signal flow diagram of this scheme (based on the block diagram of Fig. A1) is shown in Table A5. If the assumed delay from Table A4 is used to determine the timing, it becomes apparent that the signal ripples from Step 2 through Step 7 is 410 ns, irrespective of whether TOA or some other parameter is being processed. Using this information and the memory clocking scheme discussed previously yields the parallel memory data of Table A2.

Table A5 — Hardware Timing Diagram for Parallel Accessed Memory

1.*	Get $\hat{S}_m$; Get $\hat{S}_p$;	Get PRI;	Get $\hat{D}_k$
2.	Latch $\hat{S}_m$; Latch $\hat{S}_p$;	Latch PRI	Latch $\hat{D}_k$
3.	Subtract $(\hat{S}_m - \hat{S}_p)$		
4.		Add PRI	Add/Subtract $(\hat{D}_k + \lvert\hat{S}_m - \hat{S}_p\rvert / 16)$
5.	Add $(\hat{S}_p[k+1])$		Subtract $[(\hat{D}(k+1))]$
6.			Add 3σ
7.	Add (U.L.); Subtract (L.L.)		
8.*	Store U.L.; Store L.L; Store $\hat{S}_p$;	Store PRI;	Store 3σ

*Represents external memory access.

Appendix B

BENCHMARK ALGORITHMS

0. TTY Input Driver
1. Message Buffering and Transmission
2. Multiple Priority Interrupt Handler
3. Virtual Memory Exchange
4. Scale Vector Display
5. Array Manipulation — LU Decomposition
6. Target Tracking
7. Digital Communications Processing
8. Hash Table Search
9. Linked List Insertion
10. Presort on Large Address Space
11. Autocorrelate on Large Address Space
12. Character Search
13. Boolean Matrix Transpose
14. Record Unpacking
15. Vector-to-Scan Line Conversion

O. *Simple Device I/O — TTY Input Driver*

Features Tested: I/O to slow devices with minimal interfaces.

Problem: Input one line of ASCII characters from a TTY device. ASCII rubouts should delete the previous character. A carriage return terminates the line.

Algorithm: A subroutine TTYIN(BUFFER) initiates the transfer. It has a single reference parameter, the buffer to be filled. The buffer consists of

```
ADDRESS TERMADOR
CHARACTER CBUF[1:?]
```

The buffer is assumed to be large enough for the line. The transfer is started and the routine returns.

The interrupt service routine collects the line in some machine-dependent manner. The TTY interface is assumed to be a minimal one. It does the serial-to-parallel conversion, but does not do fancy things like giving blue interrupts for rubouts and green ones for carriage returns. When a carriage return is entered, the TTY input is discontinued, and a transfer to the buffer TERMADOR is made.

1. *Fast Device I/O — Buffer Queue and Transmit*

Features Tested: I/O structure for DMA devices.

Problem: Produce a program to queue message buffers and transmit them over a DMA link in FIFO order. This program need not be reentrant.

Algorithm:

```
    RECORD BUFR(ADDRESS NEXT, ADDRESS TERMADOR, INTEGER SIZE
        INTEGER DATA[1:SIZE]);
    POINTER BUFR END, START;
    ADDRESS TEMP;
    !QUEUE SUBROUTINE
    PROCEDURE QUEUE (REFERENCE BUFFER)=
    BEGIN
    IF START NEQ 0 THEN END.NEXT - ADDRESS(BUFFER) FI;
    END ⩽ ADDRESS(BUFFER);

    !QUIT IF CHANNEL ALREADY RUNNING
    IF START NEQ 0 THEN RETURN
    ELSE
       START ⩽ ADDRESS(BUFFER);
       TEMP ⩽ 0;
       GO TO RESTART
    FI;
    END;

INTERRUPT:
    BEGIN
    !INSERT CODE TO TERMINATE DEVICE TRANSFER

    TEMP ⩽ START.TERMADOR; !GET TERMINATION ADDRESS
    START ⩽ START.NEXT; RESTART:
    IF START = 0
    THEN
       GO TO TEMP
    ELSE
       ! INSERT CODE TO INITIATE DEVICE TRANSFER
    FI;
    IF TEMP = 0
    THEN RETURN
    ELSE GO TO TEMP
    FI
    END
```

2. *Multiple Priority Interrupt Handler*

Features Tested: Priority handling of devices, priority level changes.

Problem: This test program is designed to process interrupts from four devices in priority order. Upon receiving an interrupt, the processor will branch to the appropriate device service routine. All interrupts from lower priority devices will be disabled. Device priority is equal to device

number; dev 1 has lowest priority, 4 has highest. After the device-dependent service, the device I/O is added to the executive queue for user scheduling purposes.

For test purposes we are not really interested in providing device-dependent or executive service routines. However, we are interested in context swap costs. Therefore, we will use the following rather peculiar method.

Each device service routine will be simulated by the algorithm below. Prior to performing the actual measures, you will comment out the code you wrote to do the algorithm. You will not remove any context save or restore code necessitated. Your code may not make use of the fact that all the device service routines are the same.

```
!DEVICE SERVICE ROUTINE
INTEGER OWN A;
FOR 1 ≤ 1 TO A<0:2>DO
A ≤ (A * 899) MOD 123757
OD;
```

The executive service routine will be null.

The flowchart included (Fig. B1) is merely an outline of the program desired. Any hardware features that would change the flowchart without changing the function performed may be used.

3. *Virtual Memory Space Exchange*

Features tested: Executive calls and process context swap cost.

Problem: Write a supervisor call handler that provides the two functions "call" and "return." Call is function 0, return is function 1 (that is, SVC 0, SVC 1 on a 370 or 8/32, TRAP 0, TRAP 1 on a PDP-11, etc). Parameter passing is restricted to be compatible with reentrance of user routines.

The supervisor is to implement protected procedure call with parameters. "call" will select a procedure to invoke by passing an index in the range 0 leq CALLEE leq 255. This is an index into a table of address space descriptors maintained by the supervisor. PARAMETER is an address in user space. "call" performs the following functions:

1. Save the caller's state (fixed and floating access, index registers, program counter, user status registers, etc.).

2. Determine the callee's address space. The address space selected by CALLEE will contain an index of a null segment. The CALLER'S segment that contains the address PARAMETER will be inserted in this null slot. This is the parameter passing process.

3. Set up the memory mapping and protection to address the address space determined in step 2. Start the called procedure at the address specified in the address space descriptor selected by CALLEE.

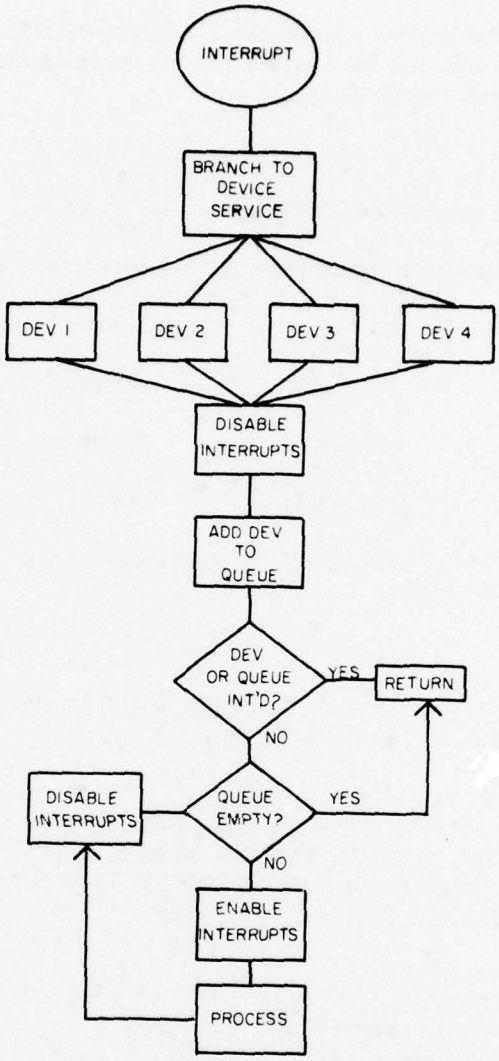


Fig. B-1—Multiple priority interrupt flowchart

In the above, "segment" means the largest unit into which the address space is divided by the mapping hardware. Thus it would be a segment on a 370, 8/32, AN/UYK-7, a page on a PDP-11, AN/UYK-28, AN/GYK-12.

"return" takes no parameters. It restores the environment active before the previous call.

Calls may be nested up to eight deep; you need not check for this bound being exceeded. Note that in nested calls the CALLER'S parameter segment is part of his state and must be

unchanged after the called routine returns. You may assume that calls will not be recursive. You need not check for this, nor for a return with no matching call. You need not build the entire address table, merely specify its format.

4. *Scale Vector Display*

Features Tested: Integer manipulation and fixed field extraction.

Problem: Scale a list of graphics vectors about a given center. The vectors are represented as

function	4 bits
X coordinate	12 bits
intensity	4 bits
Y coordinate	12 bits

If function is 0, the X,Y coordinates are displacements from the last endpoint, otherwise, X,Y is the endpoint of the vector.

Algorithm:

```
PROCEDURE SCALEADJUST(REF DLIST, VALUE LEN, VALUE XCENTR,
   VALUE YCENTR, VALUE SCALE)=
BEGIN
10 LEQ XCENTR, YCENTR LEQ 2047
!SCALE IS THE ACTUAL SCALE FACTOR TIMES 128
   INTEGER LEN, XCENTR, YCENTR, SCALE, I, XTMP, YTMP;
   RECORD VECTOR(INT4 FUNCT, INT12 X, INT4 INTEN, INT12 Y);
   VECTOR DLIST[1:LEN];

   FOR I ≤ 1 TO LEN DO
     XTMP ≤ DLIST.X[1]*SCALE;
     YTMP ≤ DLIST.Y[I]*SCALE;
     IF DLIST.FUNCT[I] = 0
     THEN
       XTMP ≤ XTMP + XCENTR*(128-SCALE);
       YTMP ≤ YTMP + YCENTR*(128-SCALE);
     FI:
     DLIST.X[I] ≤ XTMP/128;
     DLIST.Y[I] ≤ YTMP/128
   OD
   RETURN
END
```

Applications: Display generation

5. *Array Manipulation — LU Decomposition*

Features Tested: Floating-point computation, array addressing, nested iteration constructs.

Problem: Factor a square matrix into a lower and an upper triangular matrix.

Algorithm: Gaussian Elimination

```
LUDECOMP ( REFERENCE A, VALUE N) =
BEGIN
REAL ARRAY A[1:N,1:N];
REAL MULT;
INTEGER DIAG, ROW, COL;

FOR DIAG ← 1,N-1 DO
   FOR ROW ← DIAG+1,N DO
     A[ROW, DIAG] ← MULT ← A[ROW, DIAG]/A[DIAG],DIAG]
     FOR COL ← DIAG+1,N DO
       A[ROW, COL] ← A[ROW, COL] - MULT*A[DIAG, COL]
     OD
   OD
OD
END
```

Uses: Solution of linear systems

6. *Floating-Point Flow Control — Target Match*

Features Tested: Floating-point comparisons and conditional branching on floating-point results.

Problem: Given the coordinates of an unknown object, find in a table sorted by X-coordinate the closest entry. For simplicity, we will use a metric search rather than the standard Euclidean one. The distance between X1,Y1 and X2,Y2 is defined to be ABS(X1-X2)+ABS(Y1-Y2).

Algorithm: Use binary search to find the closest X coordinate and then search that neighborhood for the closest entry.

```
PROCEDURE TARGET(REFERENCE TABLE, VALUE LEN, VALUE X
        VALUE Y, REFERENCE FOUND)=
BEGIN
INTEGER LEN, START, END, MID, UP, DOWN;
REAL MINDIST;
ADDRESS FOUND;
RECORD TENTRY(REAL X, REAL Y, REAL DAT1, REAL DAT2);
TENTRY TABLE[1:LEN];

START ← 1; END ← LEN;
WHILE START < END DO
```

```
  MID ← (START+END)/2
  IF TABLE.X[MID] < X
  THEN
    START ← MID + 1
  ELSE
    END ← MID
  FI
OD;

!COMPUTE DISTANCE OF "NEAREST" X ENTRY
MINDIST ← DIST (TABLE [MID], X, Y);
FOUND ← ADDRESS(TABLE[MID]);

!SEARCH NEIGHBORHOOD FOR A NEARER ENTRY

UP ← MID + 1; DOWN ← MID - 1;
WHILE UP>0 OR DOWN > 0 DO
  IF UP > 0 THEN CHECK(UP); UP ← + 1 FI;
  IF DOWN > 0 THEN CHECK(DOWN); DOWN ← DOWN - 1 FI
OD;
RETURN;
!CHECK AN INDIVIDUAL ENTRY AGAINST CLOSEST FOUND
PROCEDURE-MACRO CHECK(J) =
BEGIN
IF J<1 OR J>LEN OR ABS(TABLE.X[J] - X) >= MINDIST
THEN J ← 0; RETURN FI;

IF DIST(TABLE[J],X,Y) < MINDIST
THEN
  MINDIST ← DIST(TABLE[J],X,Y);
  FOUND ← ADDRESS(TABLE[J])
FI;
RETURN
END
!DIST() IS THE METRIC DEFINED IN THE PROBLEM
END
```

7. Communications — Message Forwarding

Features Tested: Fast table lookup, block move.

Problem: Given a message with header that contains destination and connection I/O, place the message in the appropriate transmission line's output buffer. For each possible destination the table DESTABLE contains an entry that points to the appropriate output line table. The latter contains an entry for each connection I/O, giving the appropriate buffer to use. The message header also contains the message size in bytes including the header. The message is guaranteed to be a multiple of 4 bytes long.

Algorithm: The structure DESTABLE is assumed global.

```
PROCEDURE FORWARD (REFERENCE MSG) =
BEGIN
RECORD MESSAGE (INT16 CID, INT16 DEST, INT16 SIZE
        CHARACTER MSG[1:?]),
     BUFTABLE (INTEGER CID, ADDRESS BUFFER);
MESSAGE MSG;
POINTER BUFTABLE LINE;
INTEGER I; ADDRESS BUFFER;
EXTERNAL ADDRESS DESTABLE[1;?];

!FIND BUFFER TABLE FOR DESTINATION LINE
LINE ← DESTABLE [MSG.DEST];

!FIND RING BUFFER FOR THIS CONNECTION
I ← 1;
WHILE LINE.CID[I] NEQ MSG.CID
DO I ← I + 1 OD;
BUFFER ← LINE.BUFFER[1];

!COPY THE MESSAGE TO THE BUFFER
MOVE(ADDRESS(MSG), BUFFER, MSG.SIZE);

RETURN
END
```

8. *Hash Table Search*

Features Tested: Integer manipulation and indexing

Problem: Locate the position a key would occupy in a hash table

Algorithm:

```
    PROCEDURE HASHLOOK (REFERENCE TABLE, VALUE SIZE, VALUE KEY,
REFERENCE POSITION, REFERENCE FULL =
    BEGIN
    ADDRESS POSITION;
    INTEGER SIZE, KEY, CHECK;
    BOOLEAN FULL;
    RECORD TENTRY(INTEGER KEY, INTEGER DATA);
    TENTRY TABLE [0:SIZE-1];

    !COMPUTE FIRST PLACE TO LOOK
    CHECK ← KEY MOD SIZE;
    FULL ← FALSE;
```

```
    FOR I ← 1 TO SIZE/2 DO
      IF TABLE.KEY[CHECK] = KEY OR TABLE.KEY[CHECK] = 0
      THEN
        POSITION ← ADDRESS (TABLE. KEY[CHECK]);
        RETURN
      FI;
      CHECK ← (CHECK + 1) MOD SIZE
    OD
    FULL ← TRUE;
    RETURN
    END
```

Applications: Fast table lookup

9. *Linked List Insertion*

Features Tested: Address manipulation.

Problem: Insert a new node in an ordered doubly linked list. LISTCB contains a pointer to a list control block, containing the entries:

HEAD	pointer to first node
TAIL	pointer to last node
NUMENTRIES	number of entries in list

NEWENTRY is a pointer to a new entry to be inserted in the list. List entries have the form:

KEY	32-bit signed integer
NEXT	pointer to next entry
PREV	pointer to previous entry

The first node on the list is marked by a PREV of 0, the last by a NEXT of 0. The list shall be maintained in ascending order. The list may be empty when the routine is called; this will be indicated by a zero in NUMENTRIES.

Algorithm:

```
Procedure LISTINSERT(value LISTCB, value NEWENTRY)= Begin
  Record LCB (address HEAD, address TAIL, integer NUMENTRIES);
  Record LISTENTRY ( int32 KEY, address NEXT, address PREV);
  Pointer LCB LISTCB;
  Pointer LISTENTRY NEWENTRY, PRESENT;
  IF LISTCB.NUMENTRIES = 0
    then !list is empty, so initialize
```

```
    LISTCB.HEAD ← LISTCB.TAIL ← NEWENTRY;
    LISTCB.NUMENTRIES ← 1;
    NEWENTRY.NEXT ← NEWENTRY.PREV ← 0
    else !list not empty

    PRESENT ← LISTCB.HEAD;
    LISTCB.NUMENTRIES ← LISTCB.NUMENTRIES + 1;

     !determine position of new entry

     While NEWENTRY.KEY geq PRESENT.KEY and PRESENT.NEXT neq 0
        DO PRESENT ← PRESENT.NEXT OD;
    IF PRESENT.PREV = 0 and NEWENTRY.KEY ← PRESENT.KEY
     THEN
     !new list head

     LISTCB.HEAD ← NEWENTRY;
     NEWENTRY.PREV ← 0;
     PRESENT.PREV ← NEWENTRY;
     NEWENTRY.NEXT ← PRESENT

     else
     IF NEWENTRY.KEY geq PRESENT.KEY
       then
       !new list tail

    PRESENT.NEXT ← LISTCB.TAIL ← NEWENTRY;
    NEWENTRY.NEXT ← 0;
    NEWENTRY.PREV ← PRESENT

       else
       !insert in middle

    NEWENTRY.NEXT ← PRESENT;
    NEWENTRY.PREV ← PRESENT.PREV;
    PRESENT.PREV ← NEWENTRY;
    !back up and link with predecessor

    PRESENT ← NEWENTRY.PREV;
    PRESENT.NEXT ← NEWENTRY
   FI
  FI
FI
return end
```

10. *Manipulation of Large Address Space — Heapify*

Features Tested: Cost of address manipulation, cost of accessing large address spaces in nonsequential order.

Problem: Given an array of records in random order, rearrange them to form a HEAP.

A HEAP is a binary tree in which each node or record is GEO (alternately LEQ) its descendants.

Algorithm: The tree is compressed into an array with the root at position 1. The descendants of node I are placed at 2*1 and 2*I+1. Record 1 forms a one-element HEAP. Each record (in increasing index order) is tacked onto the end of the HEAP and the resulting set adjusted to once again be a HEAP.

```
HEAPIFY (REFERENCE REC, VALUE N)=
BEGIN
INTEGER ARRAY REC[1:N];
INTEGER CHECK, NEW;

FOR NEW ← 2,N DO
  CHECK ← NEW;
  WHILE CHECK ← NEQ 1 AND REC[CHECK] > REC[CHECK/2]
  DO

    REC[CHECK] <=> REC[CHECK/2];
    CHECK ← CHECK/2
  OD
OD
END
```

Uses: Heaps can be converted to sorted lists yielding an $N \log N$ sorting algorithm. They can also be used to implement priority queues with a cost of $\log N$ for adding or removing an entry.

11. *Sequential Access of Large Data Space — Autocorrelate*

Features Tested: Sequential access to large memory space, floating point.

Problem: Compute the autocorrelation of the vector A from 1 to T.

Algorithm:

```
PROCEDURE AUTO(REFERENCE A, VALUE N, VALUE T, REFERENCE RES)=
BEGIN
INTEGER N, T, TAU;
REAL A[1:N], RES [1:T];

FOR I ← 1 TO T DO RES[I] ← 0 OD
```

```
FOR I ← 1 TO N DO
  FOR TAU ← 1 TO T DO
    IF I+TAU-1 > N THEN EXITLOOP FI;
    RES[TAU] + A[I]*A[I+TAU-1];
  OD
OD
RETURN
END
```

12. *Character Search*

Features Tested: Byte manipulation, sequential access to byte strings.

Problem: Search a string SRCHSTR of length SRCHLNGTH to see if it contains a substring that exactly matches a string SRCHARG of length ARGLNGTH. If the search is successful the relative position of the first occurrence of the substring shall be returned. Either string may be null. A null SRCHARG shall match any string at postition 0.

Algorithm: LOC is the result returned.

```
Procedure CHARSRCH(ref SRCHSTR, value SRCHLNGTH, ref SRCHGARG,
      value ARGLNGTH, ref LOC) = Begin
  Integer I, SRCHLNGTH, ARGLNGTH;
  Bytevector SRCHSTR[0:SRCHLNGTH-1], SRCHARG[0:ARGLNGTH-1];
  LOC ← -1;
  IF ARGLNGTH eql 0 then LOC ← 0; returtn FI;
  FOR I IN 0, SRCHLNGTH-ARGLNGTH DO
    IF SRCHSTR[I:I+ARGLNGTH-I] eql SRCHARG
      then LOC ← I; return FI;
  OD;
  return END
```

13. *Boolean Matrix Transpose*

Features Tested: Random addressing and access of bits.

Problem: Given an $N \times N$ matrix of bits, compute the transpose of the matrix in place. The matrix is stored as a bit vector by rows. The bit vector begins at bit A2 of word A1.

Algorithm:

```
Procedure BMT(val N. val A1, val A2) = begin
  Integer I,J;
  Boolean [1:N,1:N] beginning at bit A2 of word A1;

  FOR I in 1,N; J in I+1,N DO
    B[I,J] <=> B[J,I]
  OD
  return
  end
```

14. *Generalized Byte Mainpulation — Unpack Record*

Features Tested: Ability to manipulate fields within words.

Problem: Design a subroutine that will unpack fields of a record into an integer array. The unpacking is controlled by a format string, which is constant at assembly time. In all cases 1 leq field size leq 32.

Algorithm: The length of each field is stored in order in the format array. If sign extend is required the value in the format is altered in some machine-dependent, distinguishable manner.

```
PROCEDURE UNPACK(REFERENCE RECORD, REFERENCE FORMAT,
      VALUE LEN, REFERENCE RESULT)=
BEGIN
BITSTRING RECORD<0:?>;
INTEGER LEN, START, RESULT[1:LEN], TEMP, I;
ARBTYPE FORMAT[1:LEN];

START ← 0;
FOR I ← 1 TO LEN DO
   TEMP ← RECORD<START:START+FORMAT[1]-1>;
   START ← START + FORMAT[1];
   IF FORMAT [I] IS A DISTINGUISHED VALUE
   THEN
     TEMP ← SIGNEXTEND(TEMP)
   FI;
   RESULT[I] ← TEMP
OD;
RETURN
END
```

Applications: Access of a data base that has been compressed for storage.

15. *Display Processing — Vector to Scan Line*

Features Tested: Integer and bit manipulation.

Problem: Given a list of vectors, produce a raster scan image. The image must be generated in top-to-bottom order to reduce internal storage; no more than one raster line may be kept in memory. The display is 1024 by 1024. Vectors are specified by their start and end coordinates and are sorted in order of their increasing starting x-coordinate.

Algorithm: The display list is converted to a more usable but larger form and stored in a temporary vector. Each raster line is generated by scanning only those vectors that appear in it.

```
PROCEDURE VECSCAN(REF DLIST, VALUE LEN, REF TEMP)=
BEGIN
RECORD DISPLAY(INT16 XS, INT16 YS, INT16 XE, INT16 YE),
   WORKLIST(INT16 XS, INT16 XE, INT32 Y, INT32 SLOPE);
```

```
DISPLAY DLIST [1:LEN];
WORKLIST TEMP[1:LEN+1];
INTEGER I, START, LINE, DENOM;
BITSTRING BIT [1:1024];

!GENERATE WORKING VECTOR
FOR I ← 1 TO LEN DO
    TEMP.XS[I] ← DLIST.XS[I];
    TEMP.XE[I] ← DLIST.XE[I];
    TEMP.Y[I] ← DLIST.YS[I]*1024;
    DENOM ← (DLIST.XE[I] - DLIST.XS[I] + 1);
    TEMP.SLOPE[I] ← (DLIST.YE[I]-DLIST.YS[I]*1024/DENOM;
OD;
TEMP.XS[I] ← 1025; !THIS MARKS LIST END
!GENERATE THE SCAN IMAGE
START ← 1;    !FIRST VECTOR TO CHECK
FOR LINE ← 1 TO 1024 DO
    BIT ← 0;    !ZAP THE WHOLE LINE
    I← START;
    WHILE TEMP.XS[I] LEQ LINE DO
    FOR K ← TEMP.Y[I]/1024 TO (TEMP.Y[I]+
            TEMP.SLOPE[I]/1024
    DO BIT [K] ← TEMP.Y[I] + TEMP.SLOPE[I];
    !FORGET THIS VECTOR IF THIS IS ITS END
    IF TEMP.XE[I] = LINE
    THEN
      TEMP[START] <=> TEMP[I];
      START ← START + 1
    FI;
    I ← I + 1
OD;
!DISPLAY BIT OD; RETURN END
```